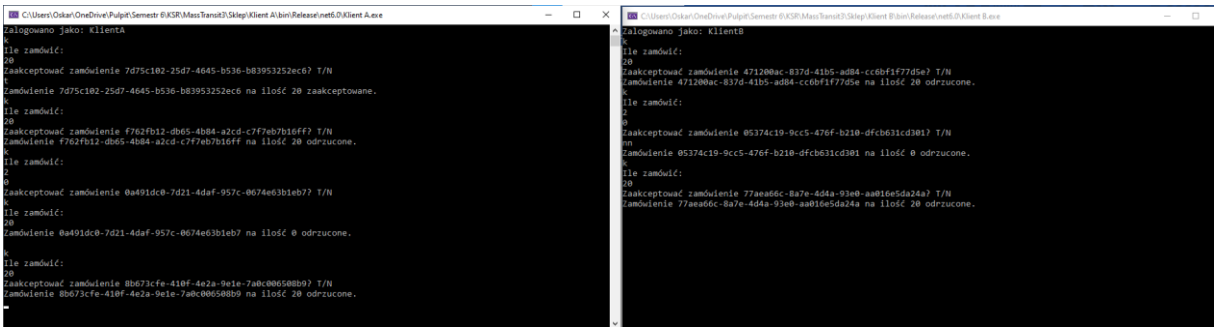
Sklep:

```
C:\Users\Oskar\OneDrive\Pulpit\Semestr 6\KSR\MassTransit3\Sklep\Sklep\bin\Release\net6.0\Sklep.exe
Sklep
Klient KlientA zamawia. Ilość: 20
Zamówienie: d8e47f3d-2e72-49e5-9fed-92b4f8ca2cc9 może być realizowane przez magazyn
Klient: KlientA potwierdzil zamowienie d8e47f3d-2e72-49e5-9fed-92b4f8ca2cc9
Klient KlientB zamawia. Ilość: 10
Zamówienie: 4fd255bc-8a3f-4bbf-9132-4852c0fd5e71 może być realizowane przez magazyn
Klient: KlientB nie potwierdzil zamowienia 4fd255bc-8a3f-4bbf-9132-4852c0fd5e71
Klient KlientA zamawia. Ilość: 10
Zamówienie: aa174452-84d3-4570-9fcc-a1d960e7d50e może być realizowane przez magazyn
Klient: KlientA potwierdzil zamowienie aa174452-84d3-4570-9fcc-a1d960e7d50e
```

```
1 odwołanie
public Sklep()
{
    InstanceState(x => x.CurrentState);

    Event(() => StartZamowienia, x => x.CorrelateBy(s => s.Login, ctx => ctx.Message.Login).SelectId(context => Guid.NewGuid()));

    Schedule(() => TO, x => x.TimeoutId, x => x.Delay = TimeSpan.FromSeconds(10));

    Initially(
        When(StartZamowienia)
            .Then(ctx =>
            {
                ctx.Instance.Login = ctx.Data.Login;
                ctx.Instance.Ilosc = ctx.Data.Ilosc;
                Console.WriteLine($"Klient {ctx.Data.Login} zamawia. Ilość: {ctx.Data.Ilosc}");
            })
            .Schedule(TO, ctx => new Timeout { CorrelationId = ctx.Instance.CorrelationId })
            .Respond(ctx => new PytanieoPotwierdzenie { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login })
            .Respond(ctx => new PytanieoWolne { CorrelationId = ctx.Instance.CorrelationId, Ilosc = ctx.Instance.Ilosc })
            .TransitionTo(Niepotwierdzone)
    );

    During(Niepotwierdzone,
        When(TimeoutEvent)
            .Then(ctx =>
            {
                Console.WriteLine($"TIMEOUT: Klient {ctx.Instance.Login} na zamowienie {ctx.Data.CorrelationId}");
            })
            .Respond(ctx => new OdrzucenieZamowienia { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login, Ilosc = ctx.Instance.Ilosc })
            .Finalize(),

        When(Potwierdzenie)
            .Then(ctx =>
            {
                Console.WriteLine($"Klient: {ctx.Instance.Login} potwierdzil zamowienie {ctx.Data.CorrelationId}");
            })
            .Unschedule(TO)
            .TransitionTo(PotwierdzoneKlient),

        When(BrakPotwierdzenia)
            .Then(ctx =>
            {
                Console.WriteLine($"Klient: {ctx.Instance.Login} nie potwierdzil zamowienia {ctx.Data.CorrelationId}");
            })
            .Respond(ctx => new OdrzucenieZamowienia { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login, Ilosc = ctx.Instance.Ilosc })
            .Finalize(),

        When(OdpowiedzWolne)
```

```
    );

    During(PotwierdzoneKlient,
        When(OdpowiedzWolne)
            .Then(ctx =>
            {
                Console.WriteLine($"Zamówienie: {ctx.Data.CorrelationId} może być realizowane przez magazyn");
            })
            .Respond(ctx => new AkceptacjaZamowienia { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login, Ilosc = ctx.Instance.Ilosc })
            .Finalize(),

        When(OdpowiedzWolneNegatywna)
            .Then(ctx =>
            {
                Console.WriteLine($"Zamówienie: {ctx.Data.CorrelationId} nie może zostać zrealizowane przez magazyn");
            })
            .Respond(ctx => new OdrzucenieZamowienia { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login, Ilosc = ctx.Instance.Ilosc })
            .Finalize()
    );

    During(PotwierdzoneMagazyn,
        When(TimeoutEvent)
            .Then(ctx =>
            {
                Console.WriteLine($"TIMEOUT: Klient {ctx.Instance.Login} na zamowienie {ctx.Data.CorrelationId}");
            })
            .Respond(ctx => new OdrzucenieZamowienia { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login, Ilosc = ctx.Instance.Ilosc })
            .Finalize(),

        When(Potwierdzenie)
            .Then(ctx =>
            {
                Console.WriteLine($"Klient: {ctx.Instance.Login} potwierdzil zamowienie {ctx.Data.CorrelationId}");
            })
            .Respond(ctx => new AkceptacjaZamowienia { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login, Ilosc = ctx.Instance.Ilosc })
            .Unschedule(TO)
            .Finalize(),

        When(BrakPotwierdzenia)
            .Then(ctx =>
            {
                Console.WriteLine($"Klient: {ctx.Instance.Login} nie potwierdzil zamowienia {ctx.Data.CorrelationId}");
            })
            .Respond(ctx => new OdrzucenieZamowienia { CorrelationId = ctx.Instance.CorrelationId, Login = ctx.Instance.Login, Ilosc = ctx.Instance.Ilosc })
            .Finalize()
    );

    SetCompletedWhenFinalized();
}
```

Klienci A i B:



```
1 odwołanie
class Message : IConsumer<IPytanieoPotwierdzenie>, IConsumer<IAkceptacjaZamowienia>, IConsumer<IOdrzucenieZamowienia>
{
    Odwołania: 0
    public Task Consume(ConsumeContext<IPytanieoPotwierdzenie> context)
    {
        if (context.Message.Login == Klient.Nazwa)
        {
            Console.WriteLine($"Zaakceptować zamówienie {context.Message.CorrelationId}? T/N");
            bool czyZaakceptowac = Console.ReadKey().Key == ConsoleKey.T;

            Console.WriteLine();
            return Task.Run(() =>
            {
                if (czyZaakceptowac)
                    context.RespondAsync(new Potwierdzenie() { CorrelationId = context.Message.CorrelationId });
                else
                    context.RespondAsync(new BrakPotwierdzenia() { CorrelationId = context.Message.CorrelationId });
            });
        }
        else
            return Task.CompletedTask;
    }

    Odwołania: 0
    public Task Consume(ConsumeContext<IAkceptacjaZamowienia> context)
    {
        if (context.Message.Login == Klient.Nazwa)
            return Console.Out.WriteLineAsync($"Zamówienie {context.Message.CorrelationId} na ilość {context.Message.Ilosc} zaakceptowane.");
        else
            return Task.CompletedTask;
    }

    Odwołania: 0
    public Task Consume(ConsumeContext<IOdrzucenieZamowienia> context)
    {
```

```
Odwołania: 0
static Task Main(string[] args)
{
    var message = new Message();

    var bus = Bus.Factory.CreateUsingRabbitMq(sbc =>
    {
        sbc.Host(new Uri("amqps://auoyxxei:QE9BIbvPWsp4wwAS-OrTIKvupwppKGok@sparrow.rmq.cloudamqp.com/auoyxxei"), h =>
        {
            h.Username("auoyxxei");
            h.Password("QE9BIbvPWsp4wwAS-OrTIKvupwppKGok");
        });
        sbc.ReceiveEndpoint("klienta", ep => ep.Instance(message));
    });

    bus.Start();
    Console.WriteLine($"Zalogowano jako: {Klient.Nazwa}");

    while (true)
    {
        if (Console.ReadKey().Key == ConsoleKey.K)
        {
            Console.WriteLine("\nIle zamówić:");

            int ilosc = 0;
            try
            {
                ilosc = Convert.ToInt32(Console.ReadLine());
                bus.Publish(new StartZamowienia() { Login = Klient.Nazwa, Ilosc = ilosc });
            }
            catch (Exception)
            {
                Console.WriteLine("Nie podano liczby!");
            }
```

Magazyn:

```
1 odwołanie
class Magazyn : IConsumer<IPytanieoWolne>, IConsumer<IAkceptacjaZamowienia>, IConsumer<IOdrzucenieZamowienia>
{
    Odwołania: 6
    public int Wolne { get; set; } = 0;
    Odwołania: 5
    public int Zarezerwowane { get; set; } = 0;

    Odwołania: 0
    public Task Consume(ConsumeContext<IPytanieoWolne> context)
    {
        return Task.Run(() =>
        {
            if (Wolne >= context.Message.Ilosc)
            {
                Wolne -= context.Message.Ilosc;
                Zarezerwowane += context.Message.Ilosc;
                Console.Out.WriteLineAsync($"Można zrealizować zamówienie {context.Message.CorrelationId} na ilość {context.Message.Ilosc}");
                context.RespondAsync(new OdpowiedzWolne() { CorrelationId = context.Message.CorrelationId });
            }
            else
            {
                Zarezerwowane += context.Message.Ilosc;
                Wolne -= context.Message.Ilosc;
                Console.Out.WriteLineAsync($"Brak wystarczającej liczby produktów do zamówienia {context.Message.CorrelationId} na ilość {context.Message.Ilosc}");
                context.RespondAsync(new OdpowiedzWolneNegatywna() { CorrelationId = context.Message.CorrelationId });
            }
        });
    }

    Odwołania: 0
    public Task Consume(ConsumeContext<IAkceptacjaZamowienia> context)
    {
        return Task.Run(() =>
        {
```

```
        return Task.Run(() =>
        {
            Zarezerwowane -= context.Message.Ilosc;
            Console.WriteLine($"Zamówienie zrealizowane: {context.Message.CorrelationId} na ilość {context.Message.Ilosc}. Przesyłka wysłana do klienta.");
        });
    }


    public Task Consume(ConsumeContext<IOdrzucenieZamowienia> context)
    {
        return Task.Run(() =>
        {
            Zarezerwowane -= context.Message.Ilosc;
            Wolne += context.Message.Ilosc;
            Console.WriteLine($"Zamówienie odrzucone: {context.Message.CorrelationId} na ilość {context.Message.Ilosc}. Produkty ponownie dostępne do sprzedaży.");
        });
    }
}

Odwołania: 0
class Program
{
    Odwołania: 0
    static void Main(string[] args)
    {
        var magazyn = new Magazyn();
        var bus = Bus.Factory.CreateUsingRabbitMq(sbc =>
        {
            sbc.Host(new Uri("amqps://auoyxxei:QE9BIbvPWsp4wwAS-OrTIKvupwppKGok@sparrow.rmq.cloudamqp.com/auoyxxei"), h =>
            {
                h.Username("auoyxxei");
                h.Password("QE9BIbvPWsp4wwAS-OrTIKvupwppKGok");
            });
            sbc.ReceiveEndpoint("magazyn", ep => ep.Instance(magazyn));
        });
```