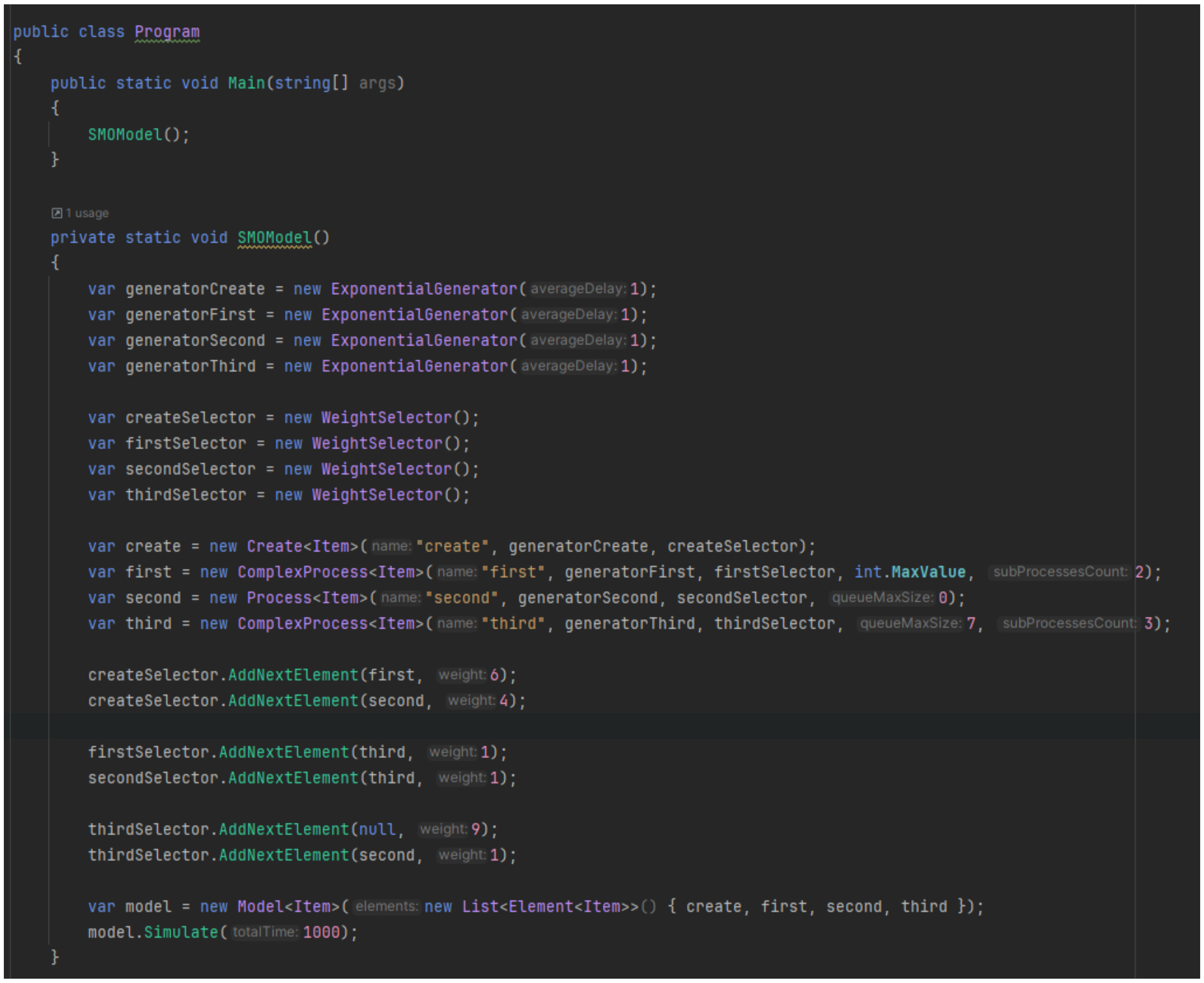
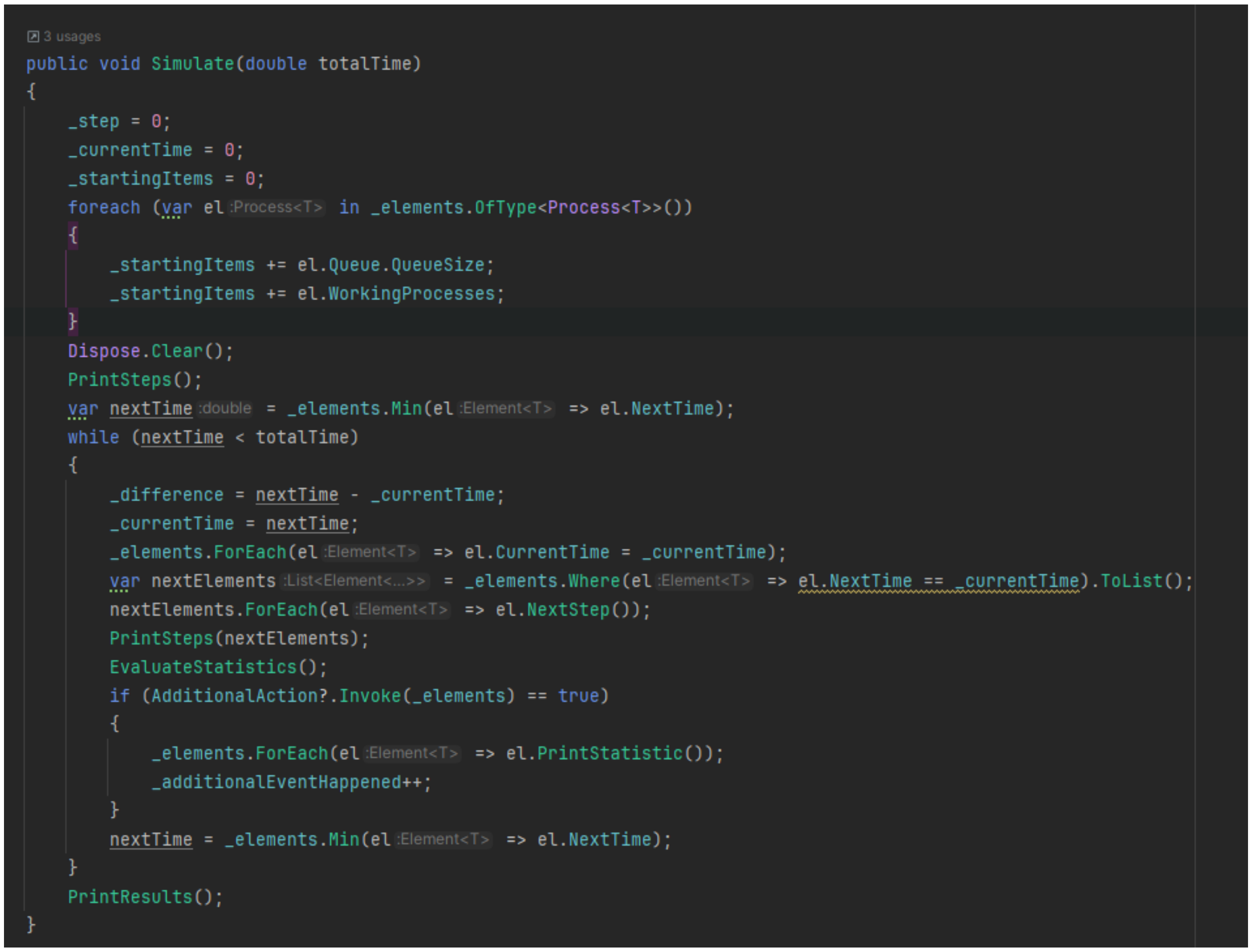
3.

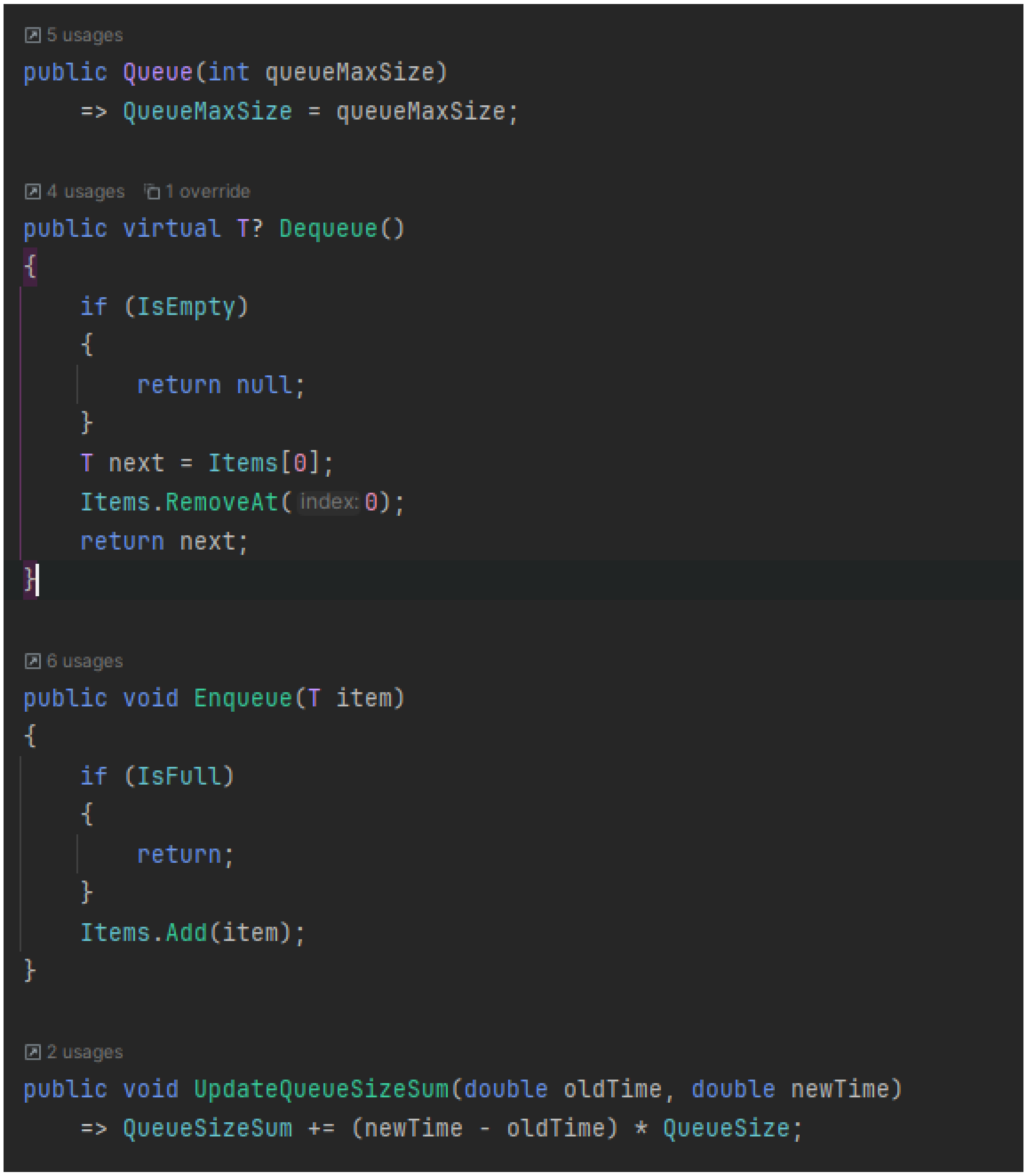


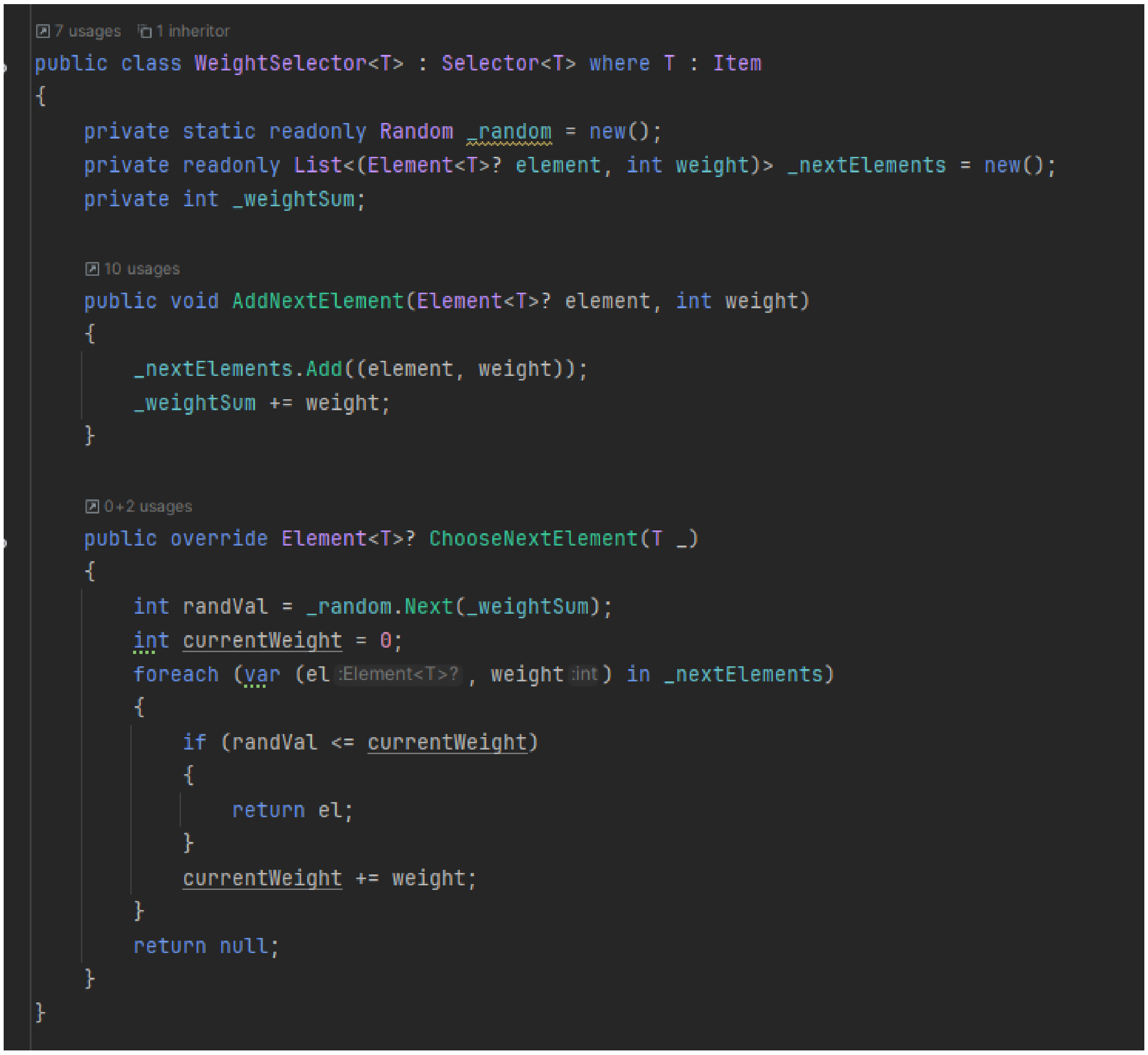
```
public class Program
{
    public static void Main(string[] args)
    {
        SMOModel();
    }

    1 usage
    private static void SMOModel()
    {
        var generatorCreate = new ExponentialGenerator( averageDelay: 1);
        var generatorFirst = new ExponentialGenerator( averageDelay: 1);
        var generatorSecond = new ExponentialGenerator( averageDelay: 1);
        var generatorThird = new ExponentialGenerator( averageDelay: 1);

        var createSelector = new WeightSelector();
        var firstSelector = new WeightSelector();
        var secondSelector = new WeightSelector();
        var thirdSelector = new WeightSelector();

        var create = new Create<Item>( name: "create", generatorCreate, createSelector);
        var first = new ComplexProcess<Item>( name: "first", generatorFirst, firstSelector, int.MaxValue,  subProcessesCount: 2);
        var second = new Process<Item>( name: "second", generatorSecond, secondSelector,  queueMaxSize: 0);
        var third = new ComplexProcess<Item>( name: "third", generatorThird, thirdSelector,  queueMaxSize: 7,  subProcessesCount: 3);

        createSelector.AddNextElement(first,  weight: 6);
        createSelector.AddNextElement(second,  weight: 4);

        firstSelector.AddNextElement(third,  weight: 1);
        secondSelector.AddNextElement(third,  weight: 1);

        thirdSelector.AddNextElement(null,  weight: 9);
        thirdSelector.AddNextElement(second,  weight: 1);

        var model = new Model<Item>( elements: new List<Element<Item>>() { create, first, second, third });
        model.Simulate( totalTime: 1000);
    }
```

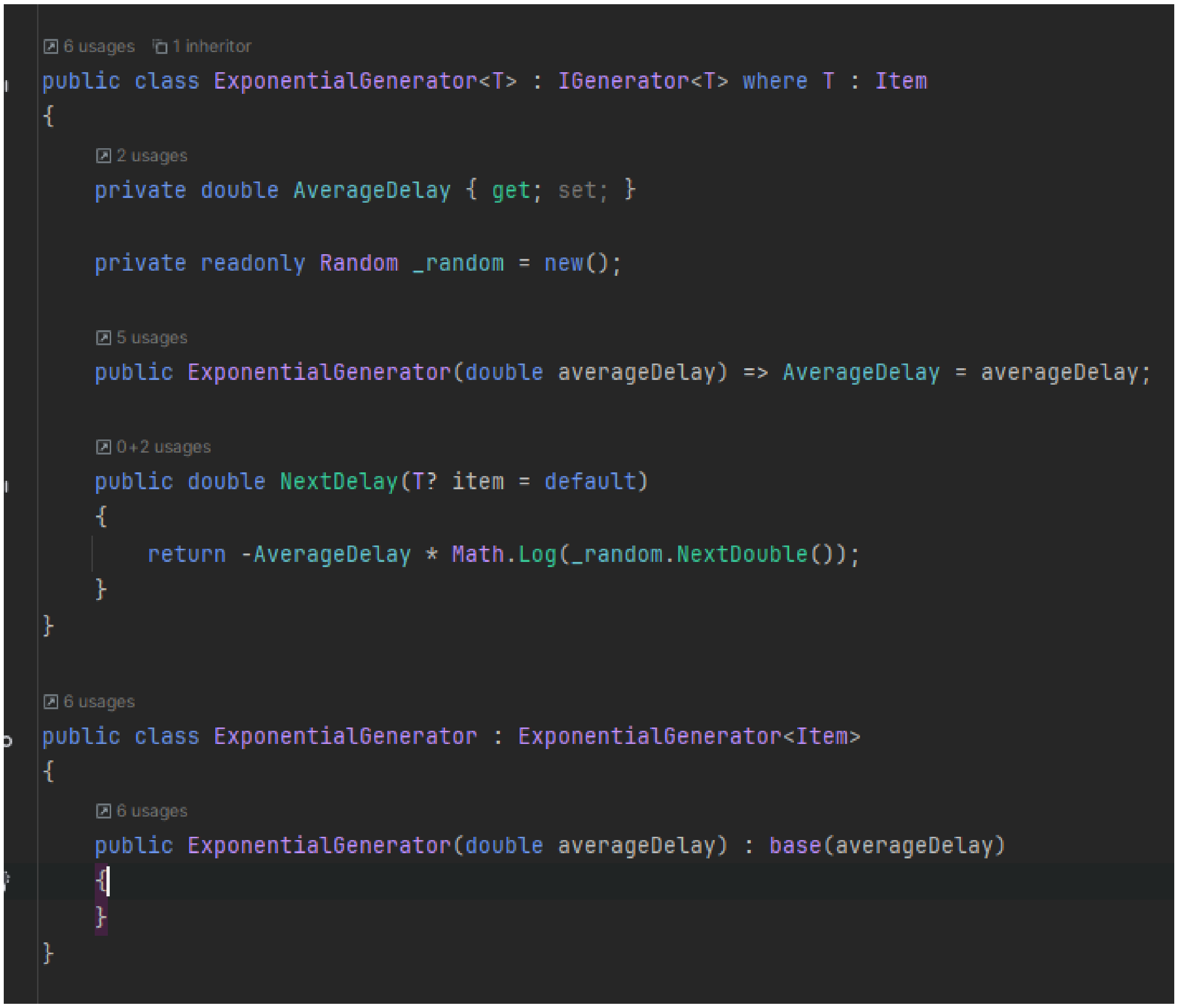
Просування в часі:


```
3 usages
public void Simulate(double totalTime)
{
    _step = 0;
    _currentTime = 0;
    _startingItems = 0;
    foreach (var el :Process<T> in _elements.OfType<Process<T>>())
    {
        _startingItems += el.Queue.QueueSize;
        _startingItems += el.WorkingProcesses;
    }
    Dispose.Clear();
    PrintSteps();
    var nextTime :double = _elements.Min(el :Element<T> => el.NextTime);
    while (nextTime < totalTime)
    {
        _difference = nextTime - _currentTime;
        _currentTime = nextTime;
        _elements.ForEach(el :Element<T> => el.CurrentTime = _currentTime);
        var nextElements :List<Element<...>> = _elements.Where(el :Element<T> => el.NextTime == _currentTime).ToList();
        nextElements.ForEach(el :Element<T> => el.NextStep());
        PrintSteps(nextElements);
        EvaluateStatistics();
        if (AdditionalAction?.Invoke(_elements) == true)
        {
            _elements.ForEach(el :Element<T> => el.PrintStatistic());
            _additionalEventHappened++;
        }
        nextTime = _elements.Min(el :Element<T> => el.NextTime);
    }
    PrintResults();
}
```

Черга, селектор і генератор:

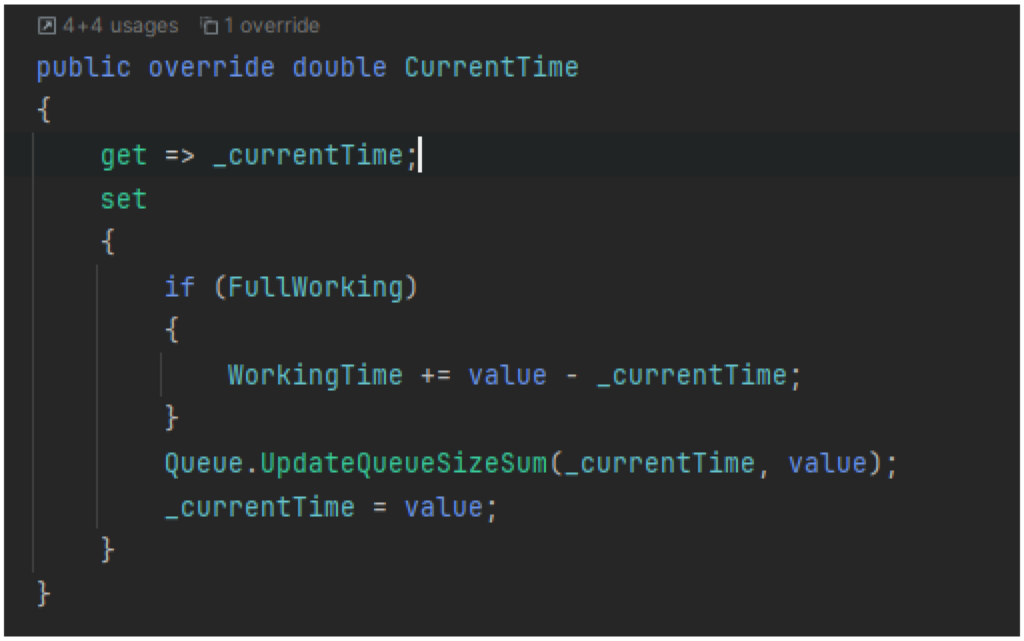

```
5 usages
public Queue(int queueMaxSize)
    => QueueMaxSize = queueMaxSize;

4 usages   1 override
public virtual T? Dequeue()
{
    if (IsEmpty)
    {
        return null;
    }
    T next = Items[0];
    Items.RemoveAt( index: 0);
    return next;
}

6 usages
public void Enqueue(T item)
{
    if (IsFull)
    {
        return;
    }
    Items.Add(item);
}

2 usages
public void UpdateQueueSizeSum(double oldTime, double newTime)
    => QueueSizeSum += (newTime - oldTime) * QueueSize;
```


7 usages 1 inheritor
```csharp
public class WeightSelector<T> : Selector<T> where T : Item
{
    private static readonly Random _random = new();
    private readonly List<(Element<T>? element, int weight)> _nextElements = new();
    private int _weightSum;

    10 usages
    public void AddNextElement(Element<T>? element, int weight)
    {
        _nextElements.Add((element, weight));
        _weightSum += weight;
    }

    0+2 usages
    public override Element<T>? ChooseNextElement(T _)
    {
        int randVal = _random.Next(_weightSum);
        int currentWeight = 0;
        foreach (var (el :Element<T>? , weight :int) in _nextElements)
        {
            if (randVal <= currentWeight)
            {
                return el;
            }
            currentWeight += weight;
        }
        return null;
    }
}
```

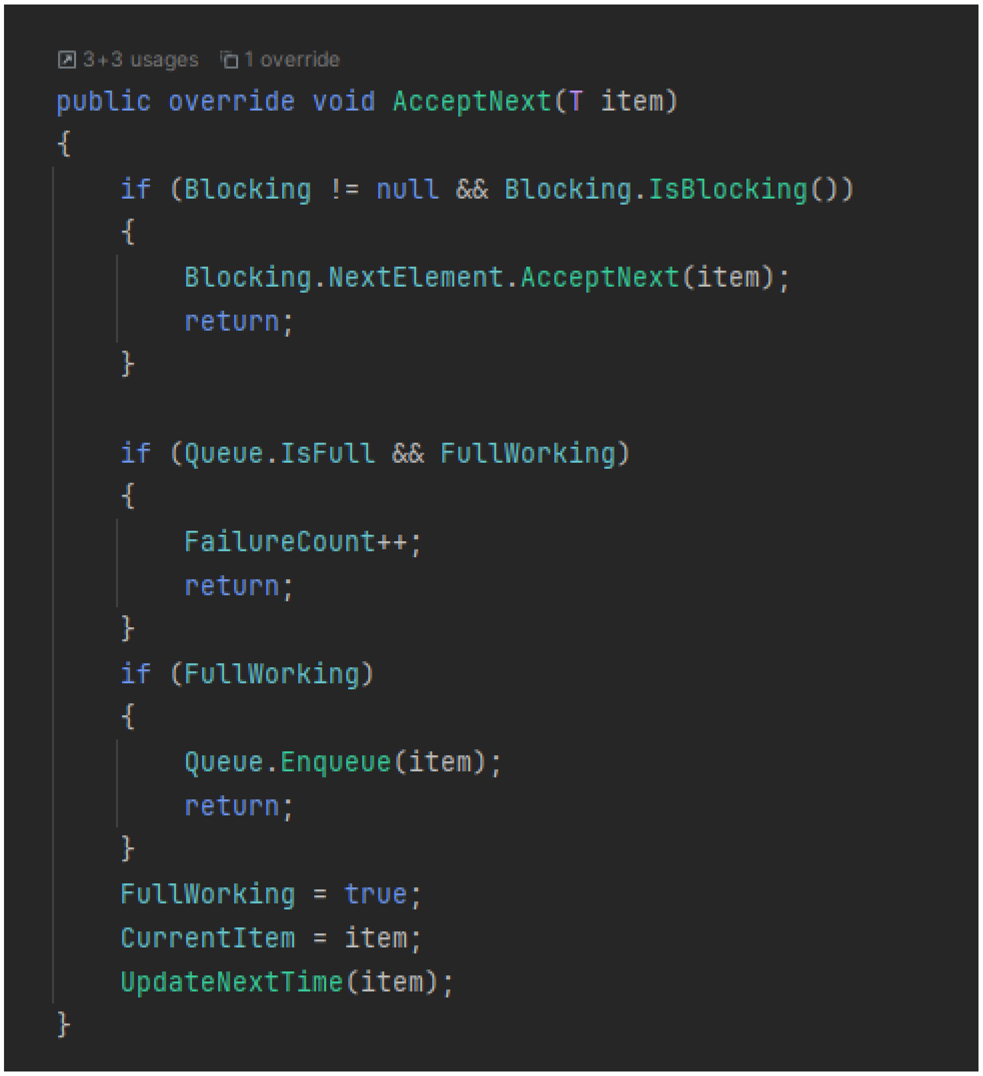


```
6 usages   1 inheritor
public class ExponentialGenerator<T> : IGenerator<T> where T : Item
{
    2 usages
    private double AverageDelay { get; set; }

    private readonly Random _random = new();

    5 usages
    public ExponentialGenerator(double averageDelay) => AverageDelay = averageDelay;

    0+2 usages
    public double NextDelay(T? item = default)
    {
        return -AverageDelay * Math.Log(_random.NextDouble());
    }
}

6 usages
public class ExponentialGenerator : ExponentialGenerator<Item>
{
    6 usages
    public ExponentialGenerator(double averageDelay) : base(averageDelay)
    {
    }
}
```

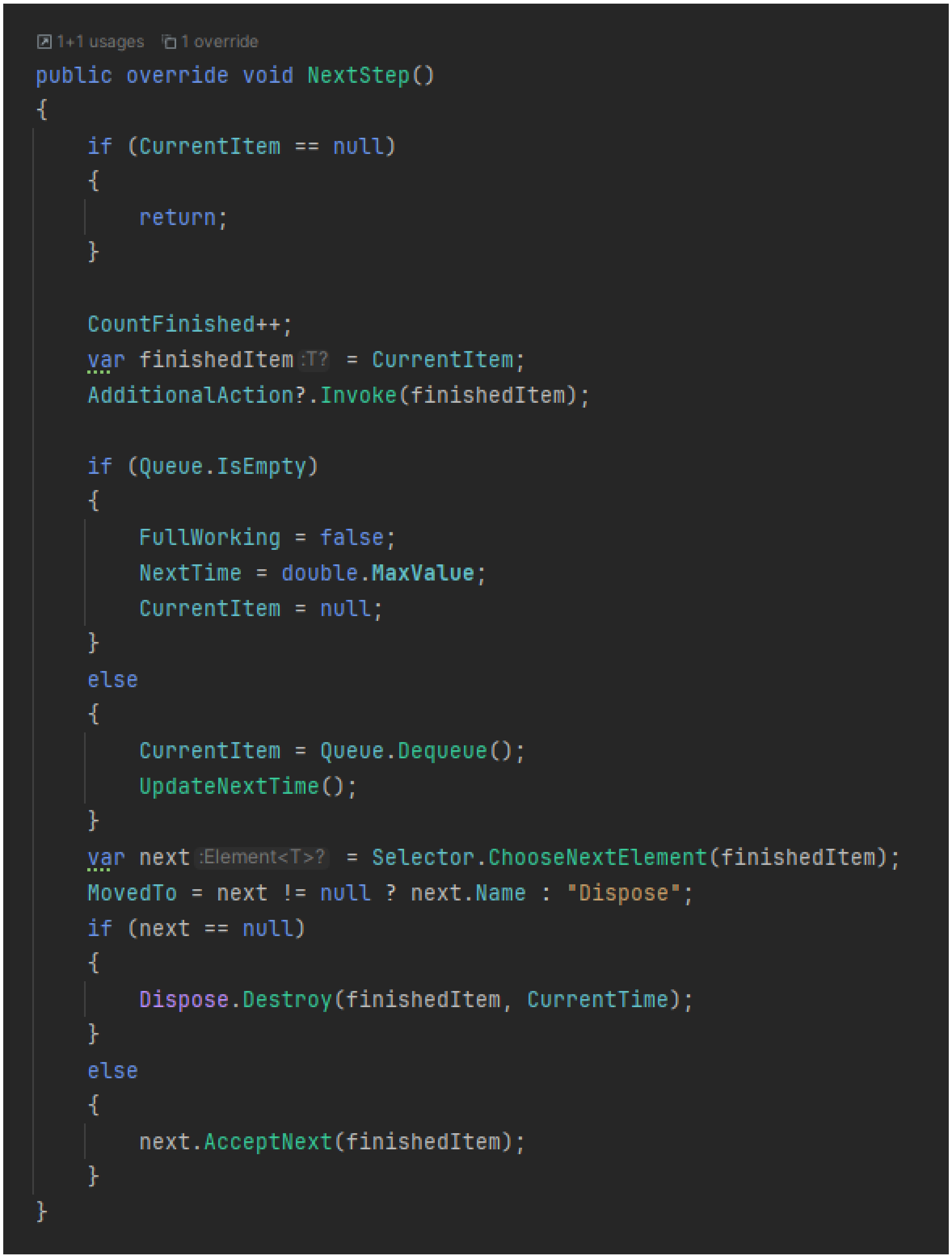
Процес:



```
4+4 usages   1 override
public override double CurrentTime
{
    get => _currentTime;
    set
    {
        if (FullWorking)
        {
            WorkingTime += value - _currentTime;
        }
        Queue.UpdateQueueSizeSum(_currentTime, value);
        _currentTime = value;
    }
}
```

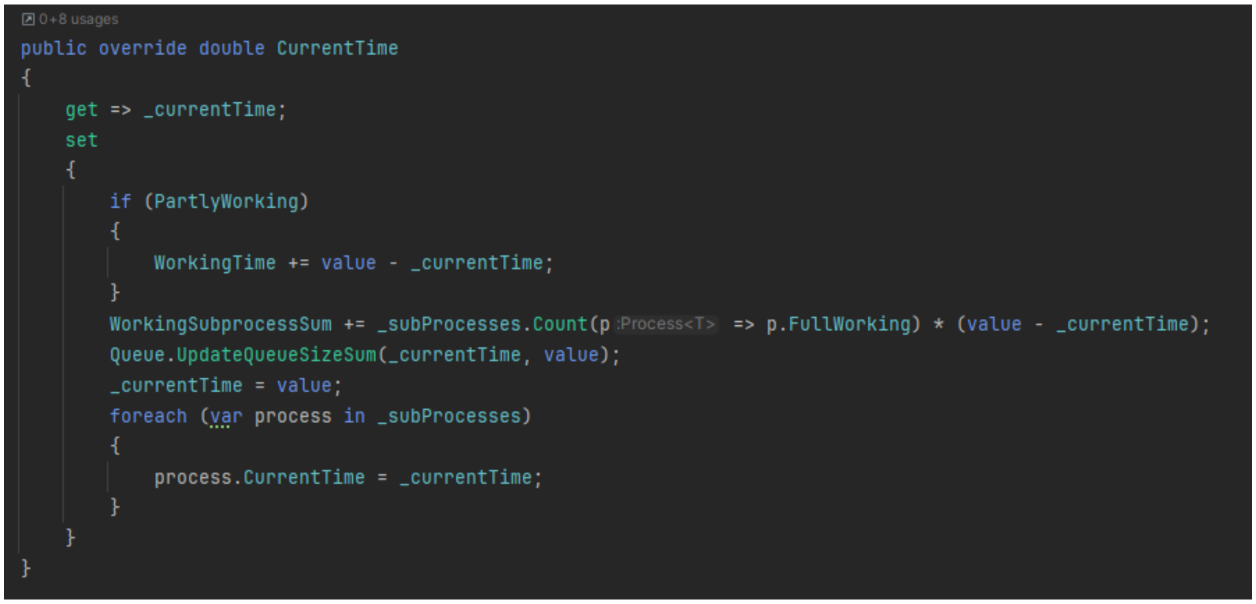


```
3+3 usages   1 override
public override void AcceptNext(T item)
{
    if (Blocking != null && Blocking.IsBlocking())
    {
        Blocking.NextElement.AcceptNext(item);
        return;
    }

    if (Queue.IsFull && FullWorking)
    {
        FailureCount++;
        return;
    }
    if (FullWorking)
    {
        Queue.Enqueue(item);
        return;
    }
    FullWorking = true;
    CurrentItem = item;
    UpdateNextTime(item);
}
```

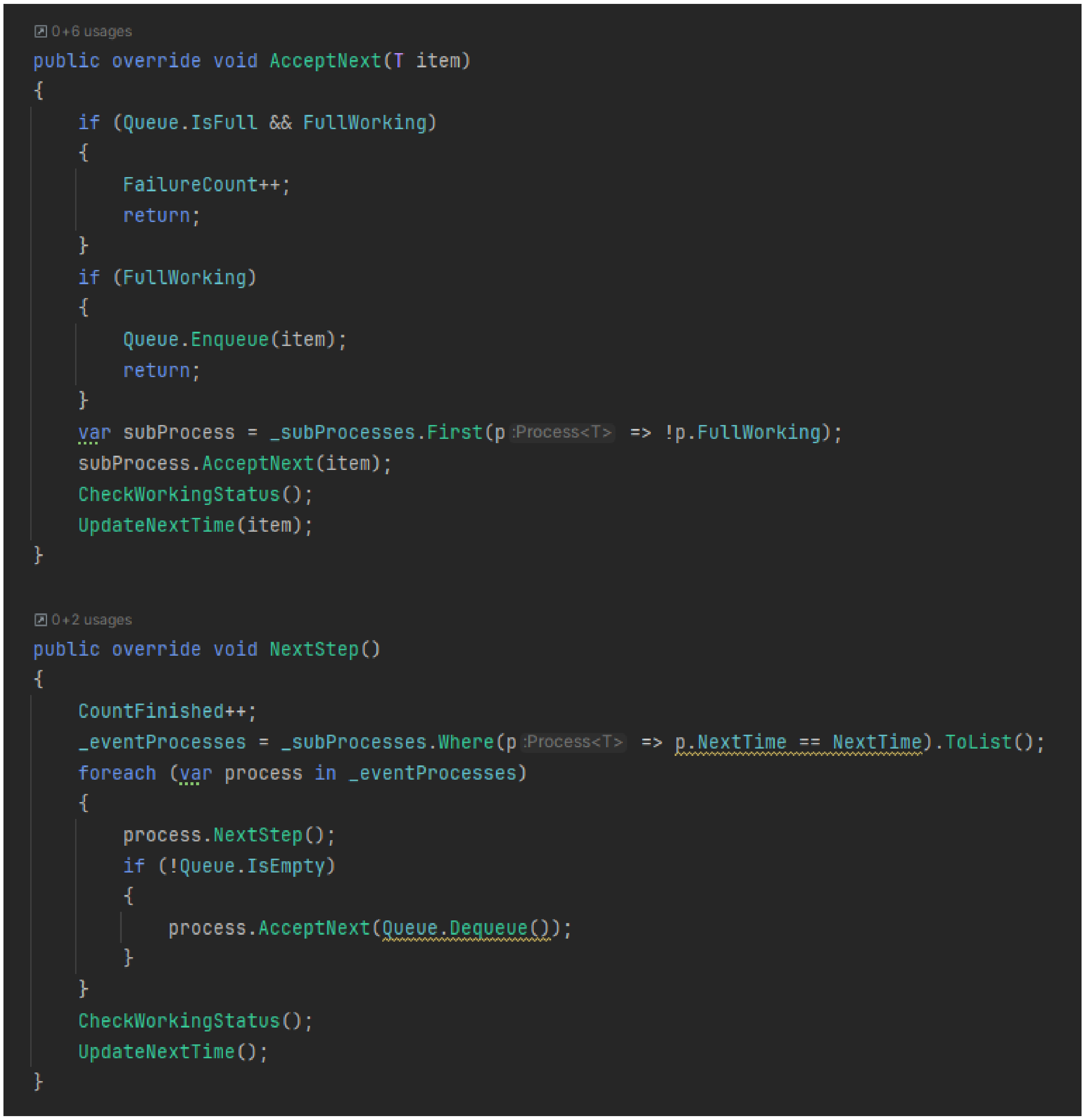


```
1+1 usages   1 override
public override void NextStep()
{
    if (CurrentItem == null)
    {
        return;
    }

    CountFinished++;
    var finishedItem :T? = CurrentItem;
    AdditionalAction?.Invoke(finishedItem);

    if (Queue.IsEmpty)
    {
        FullWorking = false;
        NextTime = double.MaxValue;
        CurrentItem = null;
    }
    else
    {
        CurrentItem = Queue.Dequeue();
        UpdateNextTime();
    }
    var next :Element<T>? = Selector.ChooseNextElement(finishedItem);
    MovedTo = next != null ? next.Name : "Dispose";
    if (next == null)
    {
        Dispose.Destroy(finishedItem, CurrentTime);
    }
    else
    {
        next.AcceptNext(finishedItem);
    }
}
```

Процес з декількома каналами:



```
public override double CurrentTime
{
    get => _currentTime;
    set
    {
        if (PartlyWorking)
        {
            WorkingTime += value - _currentTime;
        }
        WorkingSubprocessSum += _subProcesses.Count(p :Process<T>  => p.FullWorking) * (value - _currentTime);
        Queue.UpdateQueueSizeSum(_currentTime, value);
        _currentTime = value;
        foreach (var process in _subProcesses)
        {
            process.CurrentTime = _currentTime;
        }
    }
}
```



```
0+6 usages
public override void AcceptNext(T item)
{
    if (Queue.IsFull && FullWorking)
    {
        FailureCount++;
        return;
    }
    if (FullWorking)
    {
        Queue.Enqueue(item);
        return;
    }
    var subProcess = _subProcesses.First(p :Process<T>  => !p.FullWorking);
    subProcess.AcceptNext(item);
    CheckWorkingStatus();
    UpdateNextTime(item);
}

0+2 usages
public override void NextStep()
{
    CountFinished++;
    _eventProcesses = _subProcesses.Where(p :Process<T>  => p.NextTime == NextTime).ToList();
    foreach (var process in _eventProcesses)
    {
        process.NextStep();
        if (!Queue.IsEmpty)
        {
            process.AcceptNext(Queue.Dequeue());
        }
    }
    CheckWorkingStatus();
    UpdateNextTime();
}
```

4.

Див 8.3