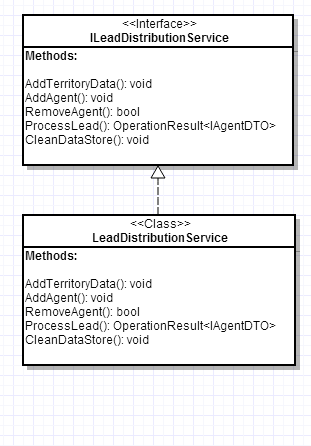
**Lead Distribution Application**

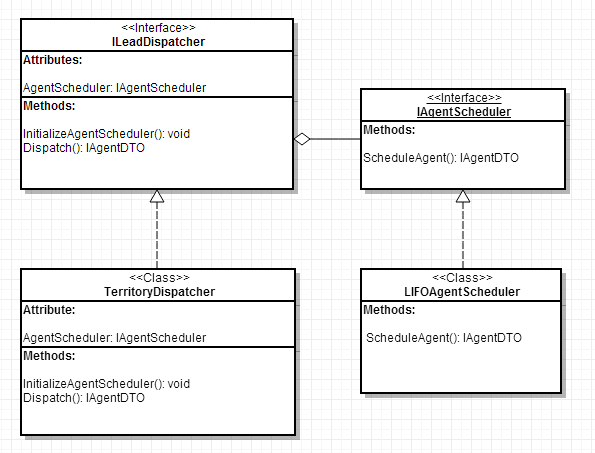
The solution consists of five class libraries and a service library. Lead Distribution problem is implemented in the service library which makes use of other class libraries.

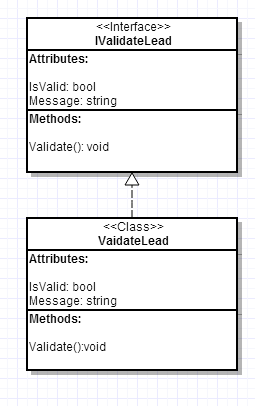
* LeadDistribution.DTO class library contains all lightweight data transfer objects which will be used across layers.
* LeadDistribution.Shared class library contains shared components which can be referenced by other class libraries.
* LeadDistribution.Shared.Test class library contains unit test classes for the components in LeadDistribution.Shared.
* LeadDistribution.Data class library provides API for fetching and updating the application data store. Currently, implementations persist all data in-memory.
* LeadDistribution.ServiceLibrary exposes contracts for managing agents and processing leads.
* LeadDistribution.ServiceLibrary.Test class library contains unit test classes for the components in LeadDistribution.ServiceLibrary.

The following class diagrams explain the association of some contracts and their implementations:

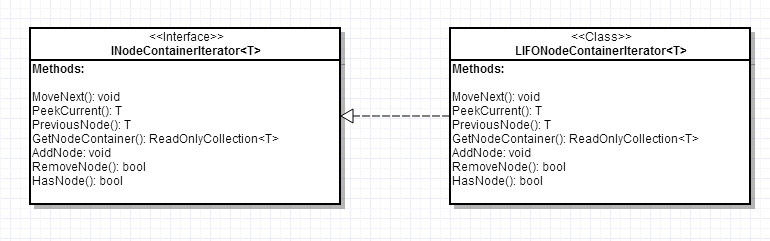
The service class “**LeadDistributionService**” implements ILeadDistributionService.

The operation contract ProcessLead() returns the operation result of AgentDTO. The given input lead is first validated, and if found valid is sent to the ILeadDispatcher. The operation result is successful only when an agent is assigned to the given lead.



The ILeadDispatcher is responsible for Dispatching the lead to the IAgentScheduler which in turn schedules the agent. TerritoryDispatcher implements the interface ILeadDispatcher. The Dispatch method of TerritoryDispatcher calls the schedule of the IAgentScheduler object (which is injected using InitializeAgentScheduler) and stores the lead and agent details in data store.

The Validation of Lead is tested by IValidateLead interface. The validate method of IValidateLead is called to validate the lead details. It sets the IsValid and Message properties according to the validation result.



The Data structure for maintaining order of agents is abstracted into INodeContainerIterator<T>. The interface provides API for iterating nodes of type T. LIFONodeContainerIterator provides implementation for INodeContainerIterator<T>. The class maintains list of nodes in form of LinkedList<T> (doubly linked list) which can easily be moved in rotational manner also facilitating addition of new nodes in the front. The following will be the time complexity for iterating nodes using LIFONodeContainerIterator<T>:

Addition: O(1)

MoveNext: O(1)

Remove: O(n)

Get all nodes: O(n)