# Agente Móvel

## Sistemas Distribuídos

**Professor Lásaro**

**Etapa 1 – Trabalho Final**

**Nomes:**
Bruno César Sales Alves
Mateus Curcino de Lima
Nathália Assis Valentim

**Uberlândia,**
**Fevereiro de 2013**

# Índice

## Agentes Móveis:

Agentes móveis são programas que têm a capacidade de mover de uma máquina para outra de uma rede durante a sua execução.

Eles podem ser utilizados com vários propósitos, dentre eles, pode-se citar os exemplos a seguir:

- Busca: onde o agente tem como finalidade, procurar um produto de um determinado preço;
- Observação: onde o agente tem como finalidade, monitorar o mercado até que surja um produto desejado;

Os Agentes móveis oferecem diversas vantagens e é possível destacar a independência de plataforma. Desde que não seja utilizado nenhum recurso exclusivo de determinada plataforma, os agentes possibilitam a integração de sistemas.

## RMI (Remote Method Invocation):

O RMI (Remote Method Invocation) é uma interface de programação, que pode ser utilizada em aplicações distribuídas desenvolvidas em Java, que permite a execução de chamadas remotas. Uma das suas vantagens seria a transparência de comunicação, considerando que, o RMI proporciona ao programador não se preocupar excessivamente com detalhes de comunicação entre elementos de um sistema, facilitando o desenvolvimento de uma aplicação.

O RMI será utilizado na realização deste trabalho a respeito de agentes móveis.

## Arquitetura do sistema

A seguir, uma simples ilustração sobre a arquitetura do sistema e o seu funcionamento:

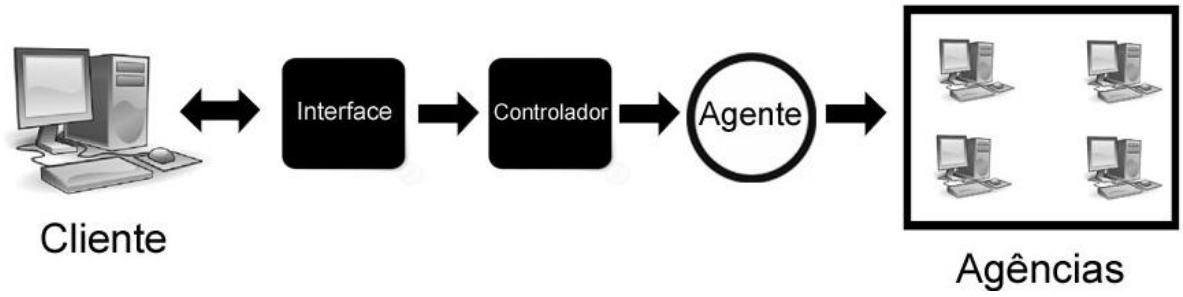



Figura 1.0 – Simples representação da arquitetura do sistema

## Descrição da arquitetura:

Exemplo: o cliente deseja realizar o serviço X, a sua comunicação com as outras partes do sistema é realizada através da interface.

O controlador, que pode ser a “agência lançadora”, tem conhecimento das agências (máquinas) existentes e quais são capazes de realizar o serviço X através de um registro (uma espécie de mapa) que diz quais serviços estão disponíveis em cada agência. Assim, o controlador fará o agente ser executado em cada agência possível.

Ao final de tudo, a interface comunicará ao cliente o resultado do serviço X.

## Infraestrutura:

O foco do trabalho é a criação de uma infraestrutura que permita a execução de quaisquer agentes que implementem uma interface em comum, no caso, “AgenteMovelInterface”.

Além disso, essa infraestrutura deve ser capaz de aceitar novos serviços e agências criadas, que devem implementar as interfaces “ServicoInterface” e “AgenciaInterface”, respectivamente.

Essas situações serão abordadas com maiores detalhes no decorrer deste relatório.

Na página a seguir, se encontra o diagrama UML de classes que representa a infraestrutura do projeto. Para exemplificar o uso da infraestrutura, usa-se situações hipotéticas, como a existência de dois serviços:

- O “ServicoX”, que implementa “ServicoXInterface”;
- O “ServicoY”, que implementa “ServicoYInterface”;

Considerando que ambos implementam “ServicoInterface”.

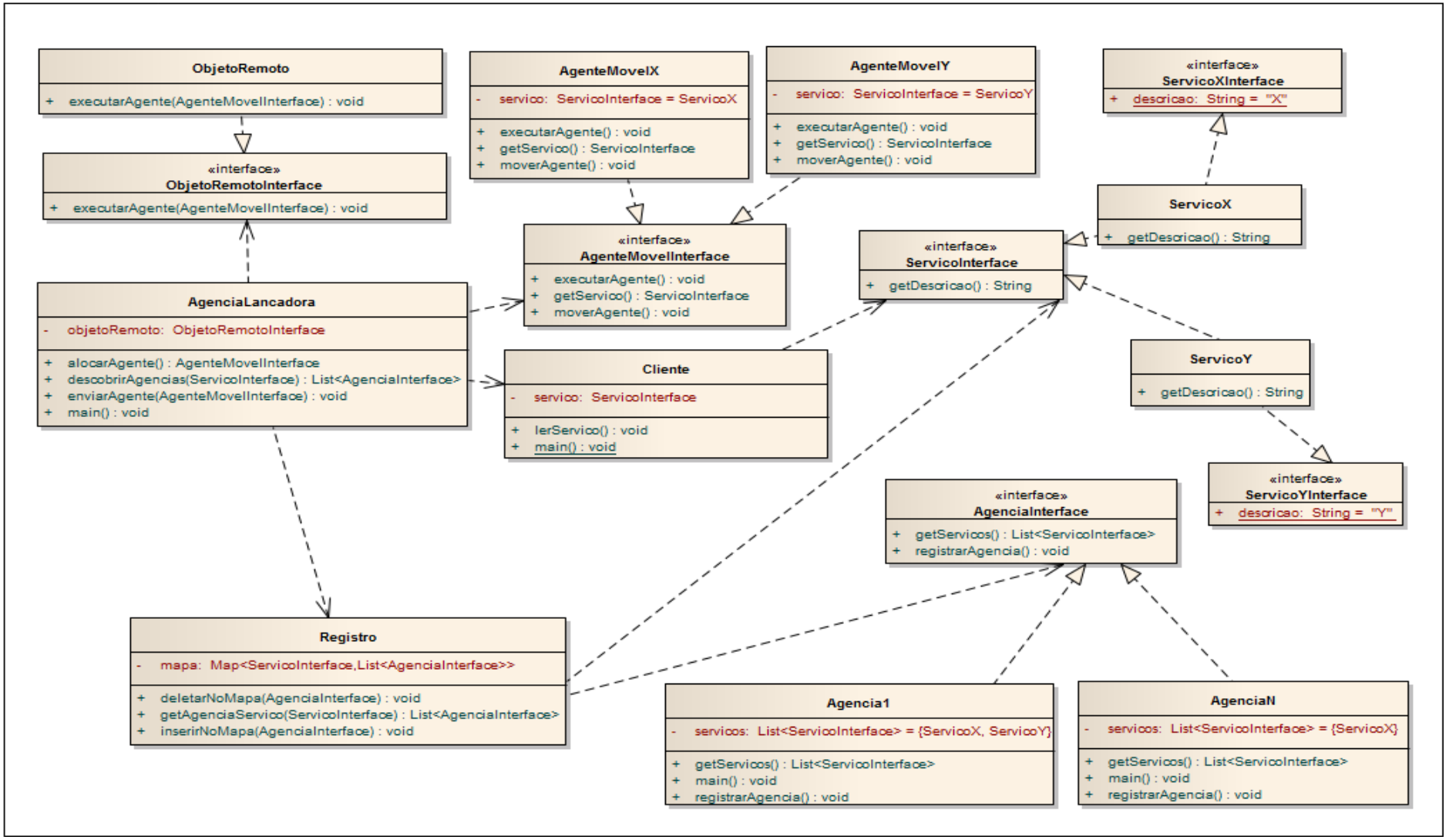
Além disso, existem dois tipos de agências que implementam “AgenciaInterface”:

- A “Agencia1”, que é capaz de realizar os serviços X e Y;
- A “AgenciaN”, que realiza somente o serviço X.

Existem dois tipos de Agentes Móveis que implementam a interface “AgenteMovelInterface”:

- O “AgenteMovelX”, que realiza o serviço X;
- O “AgenteMovelY”, que realiza o serviço Y;

## Diagrama UML de classes - infraestrutura:



## Descrição do diagrama:

**Serviços:**

A interface “ServicoInterface”, possui o método “getDescricao()”, que é responsável por retornar uma String, esse método será usado para informar a descrição dos serviços.

A interface “ServicoXInterface”, possui uma variável chamada “descricao”, do tipo String, que funciona como uma constante, sendo que as classes que implementarem essa interface, sempre terão o conteúdo “X” para a variável “descricao” (lembrando que essa interface é um exemplo hipotético para uso da infraestrutura).

A classe “ServicoX” implementará as interfaces “ServicoInterface” e “ServicoXInterface”, consequentemente terá o método “getDescricao()”, proveniente da interface “ServicoInterface”, sendo que esse método retornará o conteúdo “X”, que corresponde ao valor da variável “descricao” da interface “ServicoXInterface” (lembrando que essa classe é um exemplo hipotético para uso da infraestrutura).

A interface “ServicoYInterface”, possui uma variável chamada “descricao”, do tipo String, que funciona como uma constante, sendo que as classes que implementarem essa interface sempre terão o conteúdo “Y” para a variável “descricao” (lembrando que essa interface é um exemplo hipotético para uso da infraestrutura).

A classe “ServicoY” implementará as interfaces “ServicoInterface” e “ServicoYInterface”, consequentemente terá o método “getDescricao()”, proveniente da interface “ServicoInterface”, sendo que esse método retornará o conteúdo “Y”, que corresponde ao valor da variável “descricao” da interface “ServicoYInterface” (lembrando que essa classe é um exemplo hipotético para uso da infraestrutura).

**Cliente:**

A classe cliente é responsável pela leitura do serviço a ser executado pelo Agente Móvel. Ela possui uma variável serviço do tipo “ServicoInterface”, que será instanciada após essa leitura, no exemplo da infraestrutura, os serviços “ServicoX” e “ServicoY” são os únicos possíveis. A máquina que executar o código dessa classe, receberá alguma mensagem de retorno ao final da execução dos agentes, caso isso for necessário.

**Agências:**

A interface “AgenciaInterface” possui os seguintes métodos:

- getServicos(): responsável por retornar uma lista de serviços, do tipo “ServicoInterface”, que a agência é capaz que realizar;
- registarAgencia(): responsável por registrar as agências e seus serviços na classe “Registro”.

A classe “Agencia1”, implementará a interface “AgenciaInterface”, consequentemente todos os seus métodos. Essa classe possui uma lista do tipo “ServicoInterface”, que possui instâncias, nesse caso, de “ServicoX” e de “ServicoY”, sendo os serviços que essa agência é capaz de oferecer. Essa lista será retornada pelo método “getServicos()” proveniente da interface “AgenciaInterface” (lembrando que essa classe é um exemplo hipotético para uso da infraestrutura).

A classe “AgenciaN”, implementará a interface “AgenciaInterface”, consequentemente todos os seus métodos. Essa classe possui uma lista do tipo “ServicoInterface”, que possui uma instância de “ServicoX”, sendo o único serviço que essa agência é capaz de oferecer. Essa lista será retornada pelo método “getServicos()” proveniente da interface “AgenciaInterface” (lembrando que essa classe é um exemplo hipotético para uso da infraestrutura).

**Registro:**

A classe registro tem por finalidade registrar os serviços e as agências que são capazes de realizá-los. Esse registro é armazenado em uma espécie de mapa, que sua chave é um objeto do tipo “ServicoInterface” e seu valor é uma lista de agências (“List<AgenciaInterface>”). Possui os métodos:

- deletarNoMapa(AgenciaInterface): recebe como argumento uma agência e é responsável por retira-la do mapa nos serviços que ela é capaz de oferecer;
- getAgenciaServico(ServicoInterface): recebe como argumento um serviço e é responsável por informar uma lista de agências que são capazes de realizar tal serviço (“List<AgenciaInterface>”);
- inserirNoMapa(AgenciaInterface): recebe como argumento uma agência e é responsável por inseri-la no mapa, juntamente com os serviços que ela é capaz de oferecer.

**Agentes Móveis:**

A interface “AgenteMovelInterface” será comum para todos os agentes que utilizarem a infraestrutura do projeto, ela possui os métodos:

- executarAgente(): responsável por executar as funcionalidades de um agente com determinado serviço;
- getServico(): responsável por retornar o serviço (“ServicoInterface”) que é a tarefa do agente;
- moverAgente(): responsável por mover de uma agência para outra;

A classe “AgenteMovelX” implementará a interface “AgenteMovelInterface”, consequentemente todos os seus métodos. Essa classe possui uma variável chamada “Servico”, do tipo “ServicoInterface”, que funcionará como uma constante e, nesse caso, terá como valor

uma instância de “ServicoX”, que corresponde ao serviço que esse agente deseja realizar (lembrando que essa classe é um exemplo hipotético para uso da infraestrutura).

A classe “AgenteMovelY” implementará a interface “AgenteMovelInterface”, consequentemente todos os seus métodos. Essa classe possui uma variável chamada “Servico”, do tipo “ServicoInterface”, que funcionará como uma constante e, nesse caso, terá como valor uma instância de “ServicoY”, que corresponde ao serviço que esse agente deseja realizar (lembrando que essa classe é um exemplo hipotético para uso da infraestrutura).

**Objeto Remoto:**

A Interface “ObjetoRemotoInterface” estenderá a interface “Remote” e será implementada pela classe “ObjetoRemoto”, que implementará o seguinte método:

- executarAgente(AgenteMovelInterface): recebendo como argumento um objeto do tipo “AgenteMovelInterface”, através desse objeto invocará o método “executarAgente()”;

A classe “ObjetoRemoto”, além de implementar “ObjetoRemotoInterface” estende “UnicastRemoteObject”.

**Agência lançadora:**

A classe “AgenciaLancadora” funciona como uma espécie de controlador, pois é responsável por alocar os agentes móveis e também usa a classe “Registro” para descobrir as agências que fornecem o serviço do agente alocado, lançando-o para algumas dessas agências (os agentes serão alocados com base no serviço desejado na classe “Cliente”).

A “AgenciaLancadora” possui uma variável do tipo “ObjetoRemoto”, que pode ser utilizada para invocar o método “executarAgente()” dos agentes alocados.

Possui os seguintes métodos:

- alocarAgente(): aloca e devolve um objeto do tipo “AgenteInterface”;
- descobrirAgencias(ServicoInterface): recebe como argumento um objeto do tipo “ServicoInterface” e descobre as agências capazes de realizar tal serviço através da classe “Registro”, retornando uma lista de agências (“List<AgenciaInterface”);
- enviarAgente(AgenteMovelInterface): recebe como parâmetro um agente móvel e vai envia-lo para alguma agência descoberta através do método “descobrirAgencias(ServicoInterface)”;
- main().

## Exemplo concreto a ser utilizado:

## Propósito do Agente Móvel – descrição:

Para fazer uso dessa infraestrutura e demonstrar sua funcionalidade, será criado um agente móvel que terá como finalidade (serviço), remover os arquivos de determinada extensão de um diretório pré-estabelecido.

A extensão dos arquivos que serão removidos será previamente determinada no código, como “exe”.

Além disso, o agente terá a responsabilidade de informar o espaço livre em disco (somado de todos os computadores), antes e depois da exclusão dos arquivos.

## Justificativa:

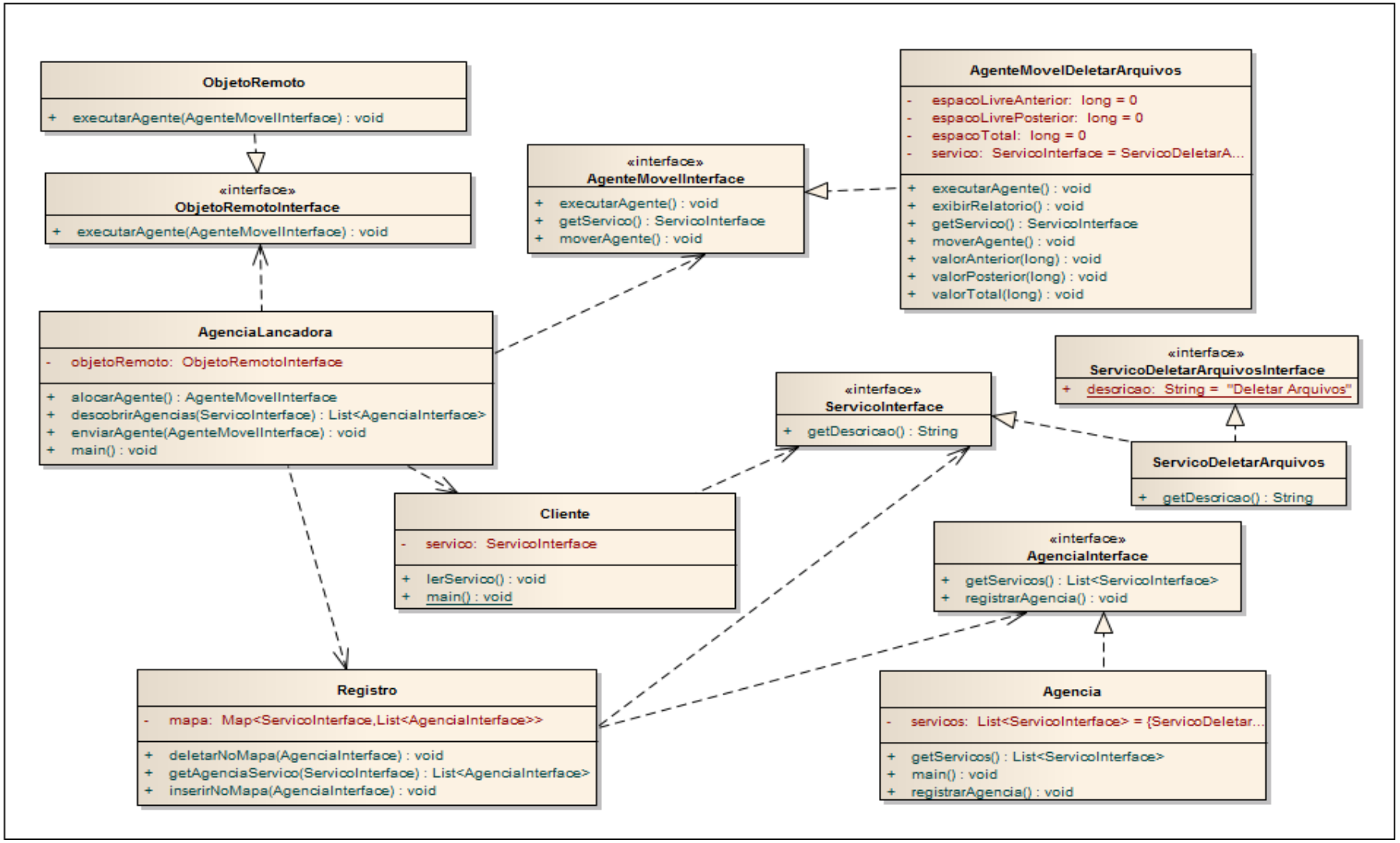
Um agente móvel com tais funcionalidades, poderá ser útil, principalmente, em ambientes de computadores compartilhados (em *lan houses,* empresas públicas e privadas), onde várias pessoas utilizam as mesmas máquinas.

Assim, será possível excluir de maneira automatizada arquivos sem grande importância para tal ambiente e ter mais espaço livre em disco para ser utilizado de maneira mais relevante.

Algumas sugestões de arquivos a serem excluídos são: arquivos do mais variados tipos em cache de navegadores e em pastas de *downloads*.

Na página a seguir, se encontra o diagrama UML de classes da infraestrutura, modificado para utilizar o agente móvel com o serviço que será utilizado no projeto.

## Diagrama UML de classes – infraestrutura + exemplo concreto:



## Descrição do diagrama:

Os detalhes comuns entre o diagrama de infraestrutura e o diagrama de infraestrutura + exemplo concreto serão omitidos durante essa descrição.

**Serviços:**

A interface “ServicoDeletarArquivosInterface”, possui uma variável chamada “descricao” do tipo String, que funciona como uma constante, com o conteúdo “Deletar Arquivos”.

A classe “ServicoDeletarArquivos” implementará as interfaces “ServicoInterface” e “ServicoDeletarArquivosInterface”, consequentemente terá o método “getDescricao()”, proveniente da interface “ServicoInterface”, sendo que esse método retornará o conteúdo “Deletar Arquivos”, que corresponde ao conteúdo da variável “descricao” da interface ServicoDeletarArquivosInterface”.

**Cliente:**

A classe cliente é responsável pela leitura do serviço a ser executado pelo Agente Móvel. Ela possui uma variável serviço do tipo “ServicoInterface”, que será instanciada após essa leitura, nesse caso, a leitura pode ser omitida e pode-se instanciar a variável diretamente para o tipo “ServicoDeletarArquivos”. A máquina que executar o código dessa classe, receberá ao final da execução do agente, um relatório contendo as informações sobre o espaço em disco antes e após a exclusão dos arquivos.

**Agências:**

A classe “Agencia”, implementará a interface “AgenciaInterface”, consequentemente todos os seus métodos. Essa classe possui uma lista do tipo “ServicoInterface”, que nesse caso, possuirá apenas uma instância de “ServicoDeletarArquivos”, sendo esse o serviço que essa agência é capaz de oferecer. Essa lista será retornada pelo método “getServicos()” proveniente da interface “AgenciaInterface”.

**Agentes Móveis:**

A interface “AgenteMovelInterface” estenderá a interface “Serializable”, considerando que o agente deverá acumular o espaço livre em disco antes e depois da remoção dos arquivos, “Serializable” será utilizado para acumular tais valores.

A classe “AgenteMovelDeletarArquivos” possui as seguintes variáveis:

- espacoLivreAnterior: tipo long, inicializado em 0, acumulará o espaço livre em disco de todos os computadores antes da exclusão dos arquivos;
- espacoLivrePosterior: tipo long, inicializado em 0, acumulará o espaço livre em disco de todos os computadores após a exclusão dos arquivos;
- espacoTotal: tipo long, inicializado em 0, acumulará o espaço em disco total de todos os computadores;
- servico: tipo “ServicoInterface”, terá uma instância de “ServicoDeletarArquivos”, sendo esse o serviço interessado pelo agente móvel.

A seguir, a descrição de todos os métodos da classe “AgenteMovelDeletarArquivos” que implementa a interface “AgenteMovelInterface”, consequentemente todos os seus métodos:

- executarAgente(): responsável por executar as funcionalidades do agente, poderá invocar alguns dos métodos descritos a seguir;
- exibirRelatorio(): quando invocado exibirá as informações a respeito do espaço em disco;
- getServico(): responsável por retornar o serviço (“ServicoInterface”) que é a tarefa do agente;
- moverAgente(): responsável por mover o agente de uma agência para outra;
- valorAnterior(long): responsável por atualizar o valor da variável “espacoLivreAnterior”;
- valorPosterior(long): responsável por atualizar o valor da variável “espacoLivrePosterior”;
- valorTotal(long): responsável por atualizar o valor da variável “espacoTotal”;

## GitHub

**Repositório no GitHub:**

https://github.com/NathaliaValentim/SD

**O repositório até o momento, além da *branche* "*master",* possui a *branche* "Trabalho":**

https://github.com/NathaliaValentim/SD/tree/Trabalho