Agente Móvel

Sistemas Distribuídos

Professor Lásaro

Etapa 1 – Trabalho Final

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

Uberlândia,
Abril 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:

- Procurar um produto de um determinado preço;
- Monitorar o mercado até que surja um produto desejado;
- Deletar arquivos em cada máquina de uma rede;
- Saber informações do espaço em disco total das máquinas de uma rede.

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 podem rodar em diferentes sistemas operacionais.

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, para a invocação de métodos remotos.

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.

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 “ServicoXInterface” e “ServicoYInterface” estendem “ServicoInterface”.

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

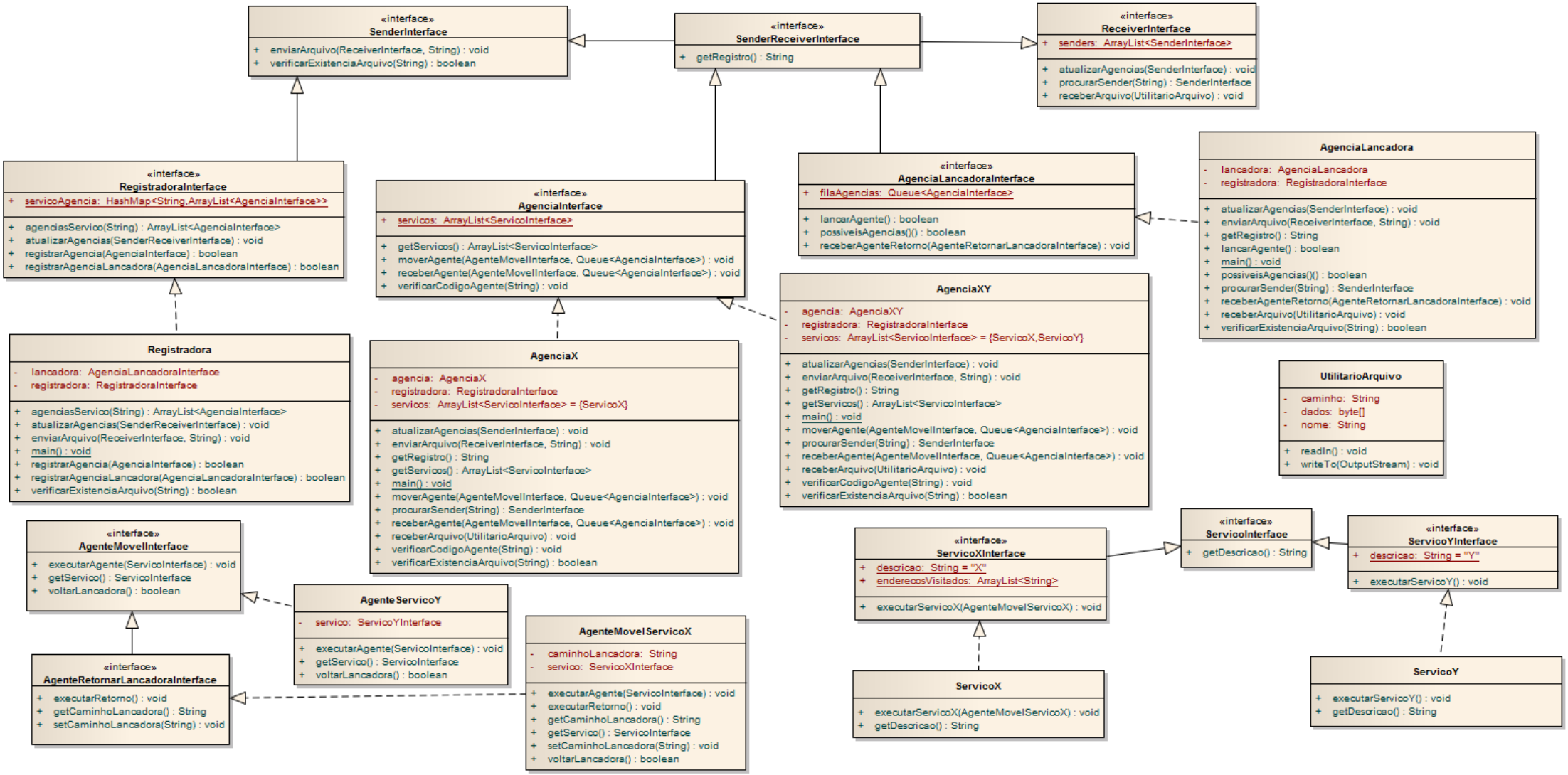
- A “AgenciaX”, que realiza somente o serviço X.
- A “AgenciaXY”, que é capaz de realizar os serviços X e Y;

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

- O “AgenteMovelServicoX”, que realiza o serviço X. Esse agente, na verdade, implementa a interface “AgenteRetornarLancadoraInterface”, que estende “AgenteMovelInterface, pois é considerado um agente, que ao fim, retorna para a agência lançadora;
- O “AgenteServicoY”, que realiza o serviço Y;

Demais classes e interfaces serão abordadas com maiores detalhes no decorrer deste relatório.

Diagrama UML de classes - infraestrutura:



OBS: As setas de dependências foram omitidas para evitar a poluição visual do diagrama.

Propósito do Agente Móvel – descrição:

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

A extensão dos arquivos que serão removidos será escolhida pelo usuário, quando executar a agência lançadora.

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:

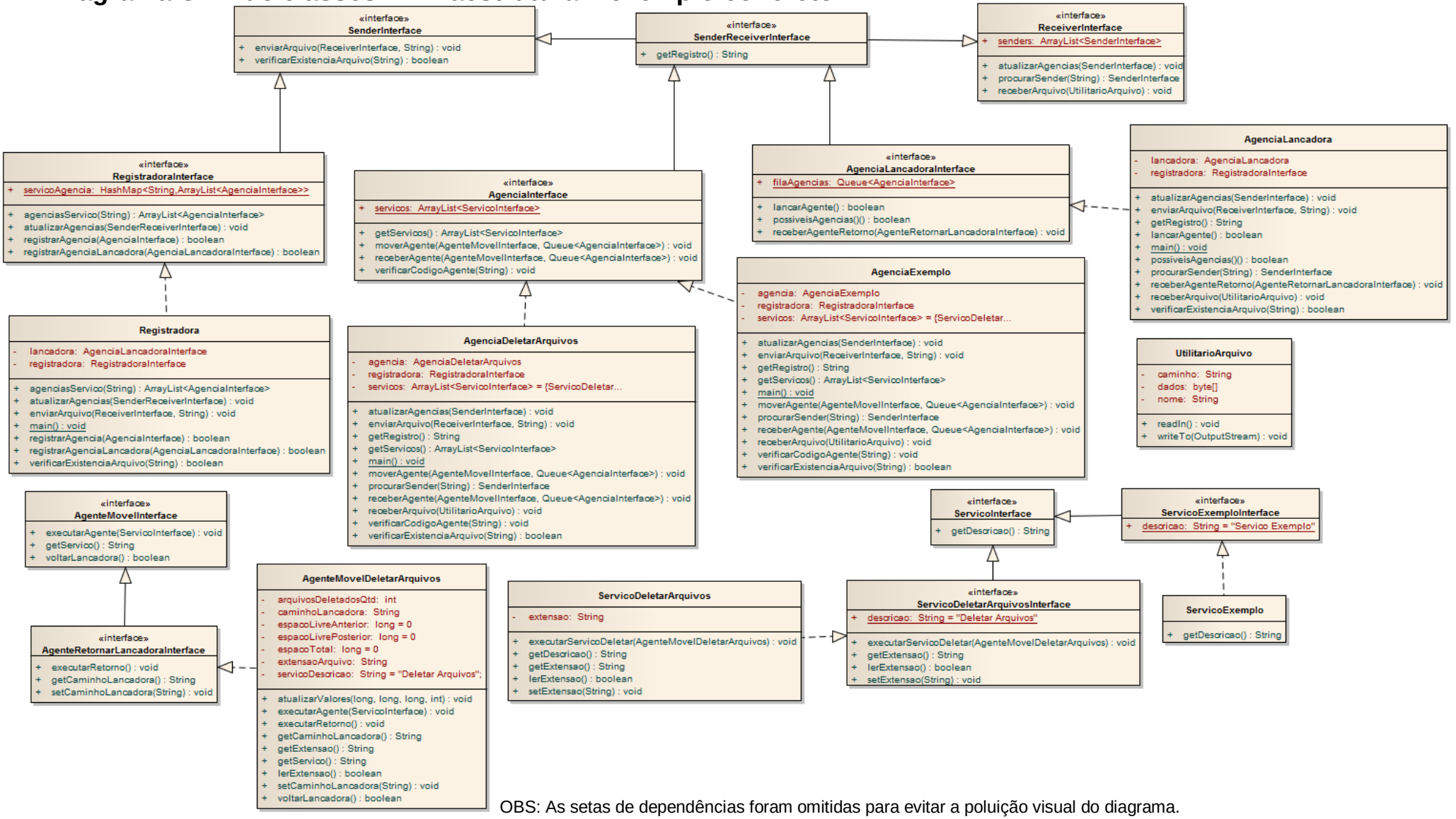
Um agente móvel com tais funcionalidades, poderá ser útil, principalmente, em ambientes de computadores compartilhados (em *lan houses,* empresas públicas e privadas), onde várias pessoas utilizam as mesmas máquinas.

Assim, será possível excluir, através do agente, 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 e as agências com o serviço que será utilizado no projeto.

Diagrama UML de classes – infraestrutura + exemplo concreto:



OBS: As setas de dependências foram omitidas para evitar a poluição visual do diagrama.

Descrição do diagrama:

As interfaces “SenderInterface” e “ReceiverInterface” estendem “Remote”, e a interface “SenderReceiverInterface” estende “SenderInterface e “ReceiverInterface”. Maiores detalhes sobre essas interfaces serão descritos na parte “Dificuldades encontradas” deste relatório.

Serviços:

A interface “ServicoInterface” possui o método “getDescricao()” que retorna uma string com a descrição de um serviço.

A interface “ServicoDeletarArquivosInterface” estende “ServicoInterface” e possui uma variável chamada “descricao” do tipo String, que funciona como uma constante, com o conteúdo “Deletar Arquivos”, além disso, o que merece destaque é o seu método “executarServicoDeletarArquivos”, que é responsável pela execução da tarefa. A interface “ServicoExemploInterface” possui uma descrição semelhante e foi criada apenas para mostrar a existência de mais de um serviço.

A classe “ServicoDeletarArquivos” implementará a interface “ServicoDeletarArquivosInterface”, consequentemente todos os seus métodos, assim como o método “getDescricao” proveniente da interface “ServicoInterface”.

Registradora:

A interface “RegistradoraInterface”, possui um “HashMap”, para guardar os serviços e as agências que disponibilizam tal serviço.

Além disso, possui os métodos:

- public boolean registrarAgencia(AgenciaInterface agencia): Insere a agência juntamente com seu serviço no “HashMap”, retorna “true” para inserção correta e “false” para inserção incorreta;
- public boolean registrarAgenciaLancadora(AgenciaLancadoraInterface lancadora): registra a agência lançadora, retorna “true” para inserção correta e “false” para inserção incorreta;
- public void atualizarAgencias(SenderReceiverInterface s): responsável por atualizar “Senders” e “Receivers”, mais detalhes na parte “Dificuldades encontradas” deste relatório;
- public ArrayList<AgenciaInterface> agenciasServico(String servico): Para um dado serviço, retorna uma lista de agências que o executam.

A classe “Registradora” implementa “RegistradoraInterface” e estende “UnicastRemoteObject”.

Agência lançadora:

A interface “AgenciaLancadoraInterface”, possui uma fila, para guardar as agências que serão visitadas pelo agente.

Além disso, possui os métodos:

- public boolean lancarAgente(): Responsável por lançar o agente para uma agência que executa o serviço desejado pelo agente móvel;
- public boolean possiveisAgencias(): Descobre as possíveis agências que executam o serviço desejado pelo agente. A registradora é utilizada para descobrir tais agências.
- public void receberAgenteRetorno(AgenteRetornarLancadoraInterface agente): Responsável por receber o retorno do agente, após percorrer todas as agências desejadas.

A classe “AgenciaLancadora” implementa “AgenciaLancadoraInterface” e estende “UnicastRemoteObject”.

Agências:

A interface “AgenciaInterface”, possui um “ArrayList”, para guardar os serviços que cada agência é capaz de executar.

Além disso, possui os métodos:

- public ArrayList<ServicoInterface> getServicos(): Retorna os serviços que a agência é capaz de executar;
- public void receberAgente(AgenteMovelInterface agenteMovel, Queue<AgenciaInterface> filaAgencias): Responsável por receber o agente e executá-lo, após isso, chama o método “moverAgente”;
- public void moverAgente(AgenteMovelInterface agenteMovel, Queue<AgenciaInterface> filaAgencias): Responsável por mover o agente para a próxima agência desejada;
- public boolean verificarCodigoAgente(String nomeAgente): Verifica se possui o código do agente antes de recebê-lo.

A classe “AgenciaDeletarArquivos” implementa “AgenciaInterface” e estende “UnicastRemoteObject”, além disso, é capaz de realizar o “ServicoDeletarArquivos”.

A classe “AgenciaExemplo” implementa “AgenciaInterface” e estende “UnicastRemoteObject”, além disso, é capaz de realizar os “ServicoDeletarArquivos” e “ServicoExemplo”.

Agente móvel:

A interface “AgenteMovelInterface” estenderá a interface “Serializable”, além disso, possui os seguintes métodos:

- public String getServico(): Retorno uma “string” com o serviço desejado pelo agente;
- public void executarAgente(ServicoInterface servico): Responsável por executar o agente, recebe como argumento um serviço, para executá-lo;
- public boolean voltarLancadora(): Informa se o agente deseja retornar para a agência lançadora através de “true” ou “false”;

A interface “AgenteRetornarLancadoraInterface” estenderá a interface “AgenteMovelnterface”, será implementa por agentes que desejam retornar para a agência lançadora. Possui os seguintes métodos:

- public void executarRetorno(): Executa o retorno do agente, como por exemplo, a impressão de informações que foram coletadas;
- public String getCaminhoLancadora(): Retorna o caminho da lançadora, ou seja, o endereço da agência lançadora no RMI;
- public void setCaminhoLancadora(String caminhoLancadora): Atualiza o caminho da lançadora.

A classe “AgenteMovelDeletarArquivos” implementa a interface “AgenteRetornarLancadoraInterface”. 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;
- servicoDescricao: tipo “String”, tem como valor “Deletar Arquivos”;
- extensaoArquivo: tipo “String”, armazenará a extensão dos arquivos a serem deletados;
- caminhoLancadora: tipo “String”, armazenará o caminho da lançadora, ou seja, o endereço da agência lançadora no RMI;
- arquidosDeletadosQtd: tipo int, inicializado em 0, armazenará a quantidade de arquivos que foi deletada.

Código do agente:

```
public interface AgenteMovelInterface extends Serializable {
	public String getServico();
	public void executarAgente(ServicoInterface servico);
	public boolean voltarLancadora();
}
public interface AgenteRetornarLancadoraInterface extends AgenteMovelInterface {
	public void executarRetorno();
	public String getCaminhoLancadora();
	public void setCaminhoLancadora(String caminhoLancadora);
}
public class AgenteMovelDeletarArquivos implements AgenteRetornarLancadoraInterface{

	private String caminhoLancadora="";//registro da registradora no RMI
	private String servicoDescricao="Deletar Arquivos";
	private String extensao=""; //extensão dos arquivos a serem deletados
	//variáveis usadas para o retorno do agente
	private long espacoLivreAnterior=0;
	private long espacoLivrePosterior=0;
	private long espacoTotal=0;
	private int arquivosDeletados=0;

	//construtor
	public AgenteMovelDeletarArquivos(){
		//chama o método de leitura da extensão dos arquivos
		while(!lerExtensao()){
			System.out.println("\nNao foi possivel ler a extensao!\n\nRepita o
			procedimento!");
		}
	}

	//Responsável pela leitura dos arquivos a serem deletados
	public boolean lerExtensao() {
		String entrada = "";
		System.out.println("\nDigite a extensao dos arquivos a serem removidos:   ");
		BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
		System.out.print("=> ");
		try {
			entrada = in.readLine();
		} catch (IOException e) {
			return false;
		}
		if (entrada == null || entrada.length() == 0)
			return false;

		this.extensao = entrada;
		return true;
	}

	//execução do agente - recebe como argumento um servico e faz o cast
	public void executarAgente(ServicoInterface servico) {
		ServicoDeletarArquivosInterface servicoDeletar =
		(ServicoDeletarArquivosInterface) servico;
		servicoDeletar.executarServicoDeletar(this);
	}

	// Relatório de retorno do agente
	public void executarRetorno() {
		System.out.println("\n***
		_____ ***\n");
		System.out.println("Resumo do servico: \n");
		System.out.println("Quantidade de arquivos deletados: "+ arquivosDeletados);
```

```
                System.out.println("Espaco total em disco (somando todas as maquinas): "
                + espacoTotal + " MB");
                System.out.println("Espaco livre (somando todas as maquinas - antes da exclusao):
                "+ espacoLivreAnterior + " MB");
                System.out.println("Espaco livre (somando todas as maquinas - apos a exclusao): "
                + espacoLivrePosterior + " MB");
                System.out.println("Total deletado: "
                + (espacoLivrePosterior - espacoLivreAnterior) + " MB");
                System.out.println("\n***
                ____________________________________________________________ ***\n");

        }

        // Atualizar valores das variáveis
        public void atualizarValores(long espacoLivreAnterior,
                        long espacoLivrePosterior, long espacoTotal, int arquivosDeletados) {
                this.espacoLivreAnterior += espacoLivreAnterior;
                this.espacoLivrePosterior += espacoLivrePosterior;
                this.espacoTotal += espacoTotal;
                this.arquivosDeletados += arquivosDeletados;
        }

        // Esse agente retornará para a lançadora
        public boolean voltarLancadora() {
                return true;
        }

        public String getServico() {
                return servicoDescricao;
        }

        public String getExtensao(){
                return extensao;
        }

        public String getCaminhoLancadora() {
                return caminhoLancadora;
        }

        public void setCaminhoLancadora(String caminhoLancadora) {
                this.caminhoLancadora = caminhoLancadora;
        }

}
```

Dificuldades encontradas:

A primeira dificuldade encontrada foi à execução do projeto em mais uma máquina, onde se rodava, por exemplo, a agência registradora, as agências e a agência lançadora em máquinas diferentes. A resolução encontrada consistiu em registrar cada agência com o seu endereço de ip como “host”.

A dificuldade que merece mais destaque foi lidar com a possível ausência do código do agente (arquivo “.class”) nas agências. Para resolver tal situação, foram criadas as interfaces: “SenderInterface”, “ReceiverInterface” e “SenderReceiverInterface”. Todas as classes que estenderem “AgenciaInterface” e “AgenciaLancadoraInterface” serão “SenderReceiver”, e a “Registradora” será “Sender”.

“Receivers” possuem uma lista de “Senders”, cabe a “Registradora” a função de atualizar as listas de “Senders” para cada novo “Receiver” registrado.

Antes de receber o agente, verifica-se a existência do seu código, constatada a ausência, a agência procurará algum “sender” que possui arquivo “.class”, sendo encontrado será enviado, não sendo encontrado, o programa se encerrará. Para auxiliar no envio do arquivo existe a classe “UtilitarioArquivo”.

Essa solução foi planejada, para evitar um ponto único de falha para o envio de arquivos, sendo possível existir vários “Senders” no sistema.

GitHub:

Repositório no GitHub:

https://github.com/NathaliaValentim/SD

O repositório possui duas *branches:*

- **"*master":*** contendo o relatório;
  https://github.com/NathaliaValentim/SD/
- **"CodigoProjeto":** contendo todo o código fonte utilizado no desenvolvimento do trabalho.
  https://github.com/NathaliaValentim/SD/tree/CodigoProjet