DSID – Desenvolvimento de Sistemas de Informação Distribuídos
Escola de Artes, Ciências e Humanidades da Universidade de São Paulo
Bacharelado em Sistemas de Informação

PROF. DR. NORTON TREVISAN ROMAN

TSUYOSHI YODOGAWA

RELATÓRIO

SISTEMA DISTRIBUÍDO COM USO DO RMI

São Paulo
2021

SUMÁRIO

1. **APRESENTAÇÃO**

O programa tem a finalidade de fazer com que um programa Cliente se conecte a um programa Servidor e requisite serviços a este. Para isto é necessário a conexão entre ambos, feita através da biblioteca RMI do Java, para a execução remota do programa.

Os serviços disponíveis são:
– Troca de servidor através da porta;
– Consultar o nome do repositório corrente;
– Consultar a quantidade de peças existentes no repositório corrente;
– Inserção de novas peças no repositório do servidor conectado;
– Remoção de uma peça do repositório corrente;
– Busca de uma peça existente no repositório através de seu ID;
– Verificar o nome da peça;
– Verificar a que repositório a peça corrente pertence;
– Visualizar a descrição da peça;
– Adicionar peças como subcomponente da peça corrente;
– Visualizar quantos subcomponentes a peça corrente possui para cada subcomponente;

O programa foi testado e executado nos ambientes: Windows 10, Ubuntu 20.04 LTS e Linux Mint 20.1 Ulyssa Cinnamon.

2. **DESCRIÇÃO DO PROJETO**

O projeto é composto por 5 pacotes e 18 arquivos, onde estão organizados da seguinte forma:

– Client
- Client.java

– GUI
- GUIAlerts.java
- GUIClient.fxml
- GUIClientController.java
- GUIInsertPart.fxml
- GUIInsertPartController.java
- Main.java

– PackageServer
- RunServer.java
- Server.java

– Repositories
- RepositoryHandler.java
- Repo S1.txt
- Repo S2.txt
- Repo S3.txt
- Repo S4.txt

– ServerAPIs
- Part.java (Interface)
- PartConcrete.java
- PartRepository.java (Interface)
- PartRepositoryConcrete.java

3. ESPECIFICAÇÕES

CLIENT

Possui a classe Client.java que faz a chamada do método main da classe principal Main.java localizada no pacote GUI. O uso detalhado desta classe será descrito posteriormente.

GUI

GUIAlerts.java é uma classe responsável por gerar janelas de alertas, seja para um tratamento de erro ou para a confirmação de operação realizada pelo cliente, como a adição de subcomponente de uma peça agregada. Esta classe nos dá a possibilidade de fazermos o reúso da mesma para contextos diferentes, nos dando a possibilidade de personalizarmos o conteúdo da caixa de mensagem assim como ícone para erro ou confirmação. Abaixo seguem as imagens dos alertas para tratamento dos erros assim como para confirmações das operações realizadas pelo Cliente.

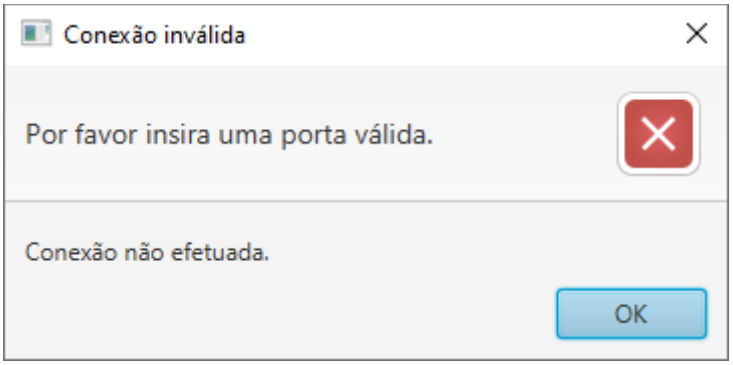

Figura 1: Conexão inválida: Porta não inserida.

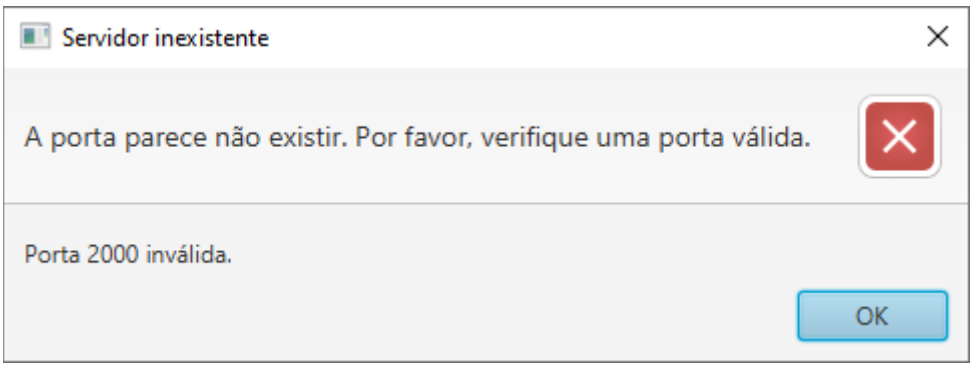

Figura 2: Servidor inexistente: Inserção de uma porta inválida.

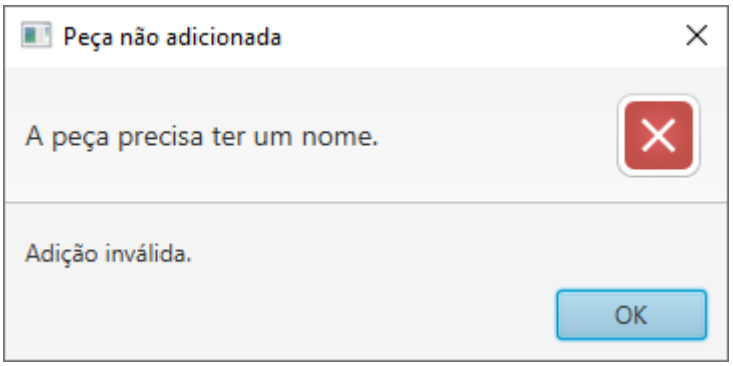

Figura 3: Peça não adicionada: Peça sem nome.

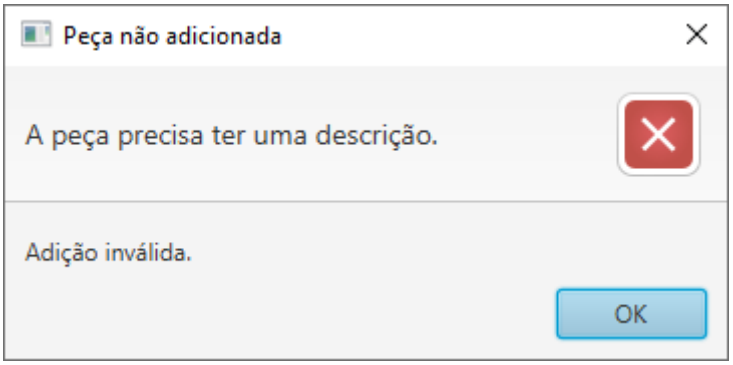

Figura 4: Peça não adicionada: Peça sem descrição.

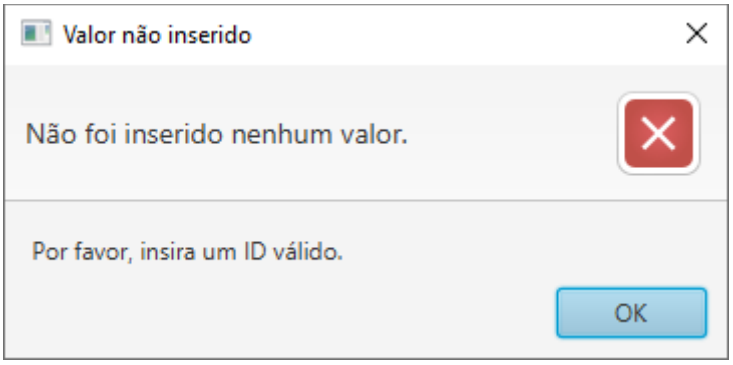

Figura 5: Busca não realizada: Valor não inserido.

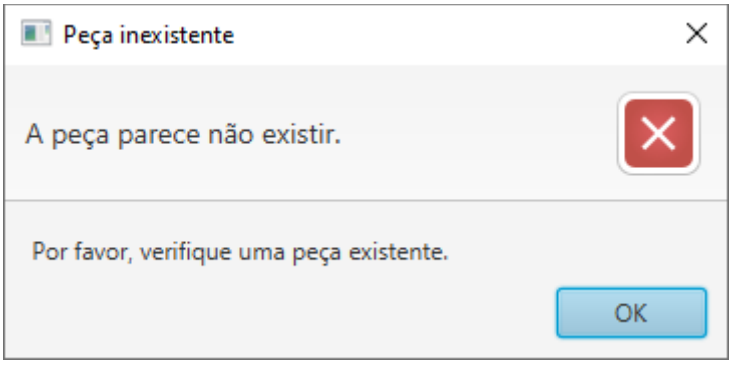

Figura 6: Busca não realizada: Valor inserido inexistente.

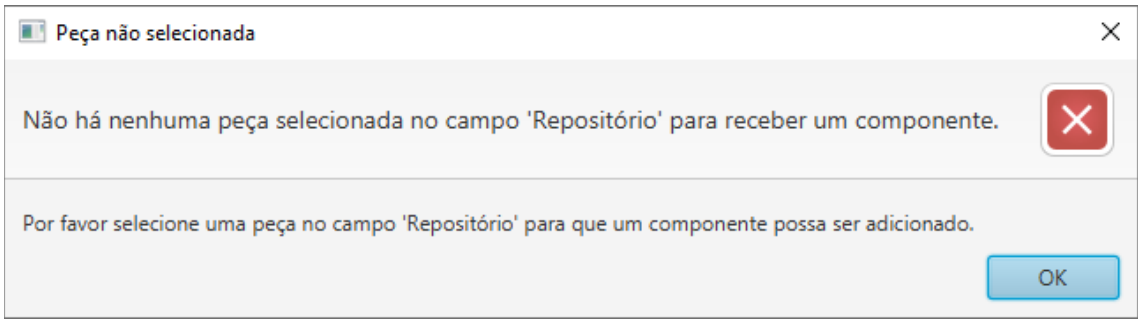

Figura 7: Peça para adição de subcomponentes não selecionada.

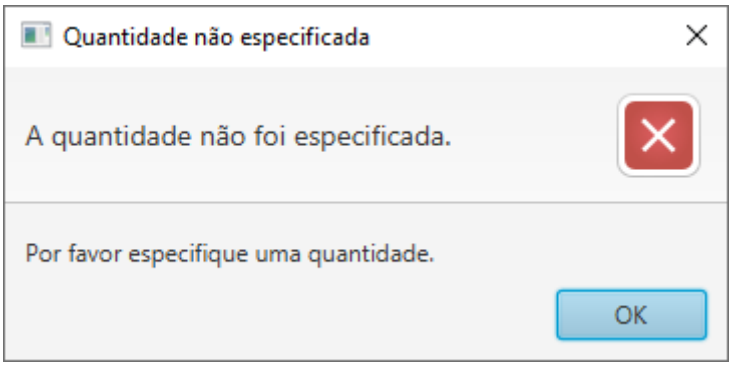

Figura 8: Quantidade de adição de subcomponentes não especificada.

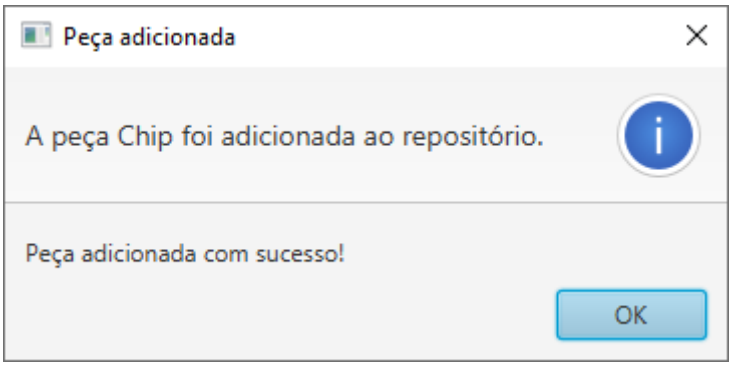

Figura 9: Confirmação de adição de peça no repositório.

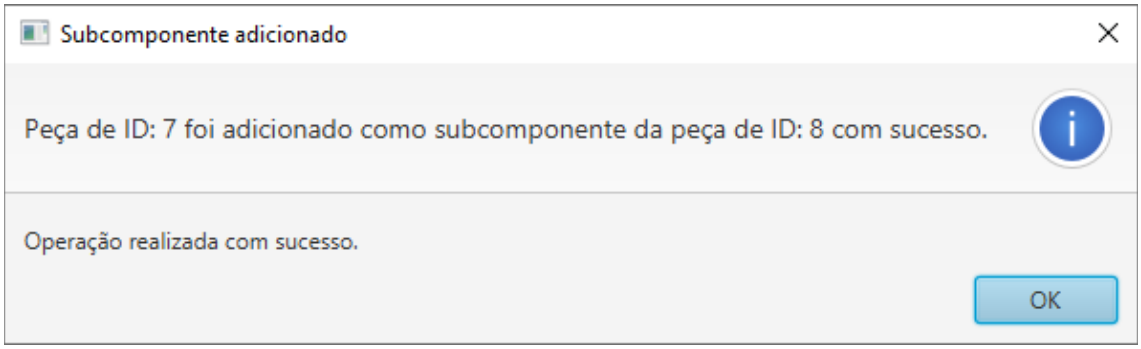

Figura 10: Confirmação de subcomponente agregado a peça corrente.

GUIClient.fxml é o arquivo que contêm os componentes da janela de interação do cliente com o servidor.

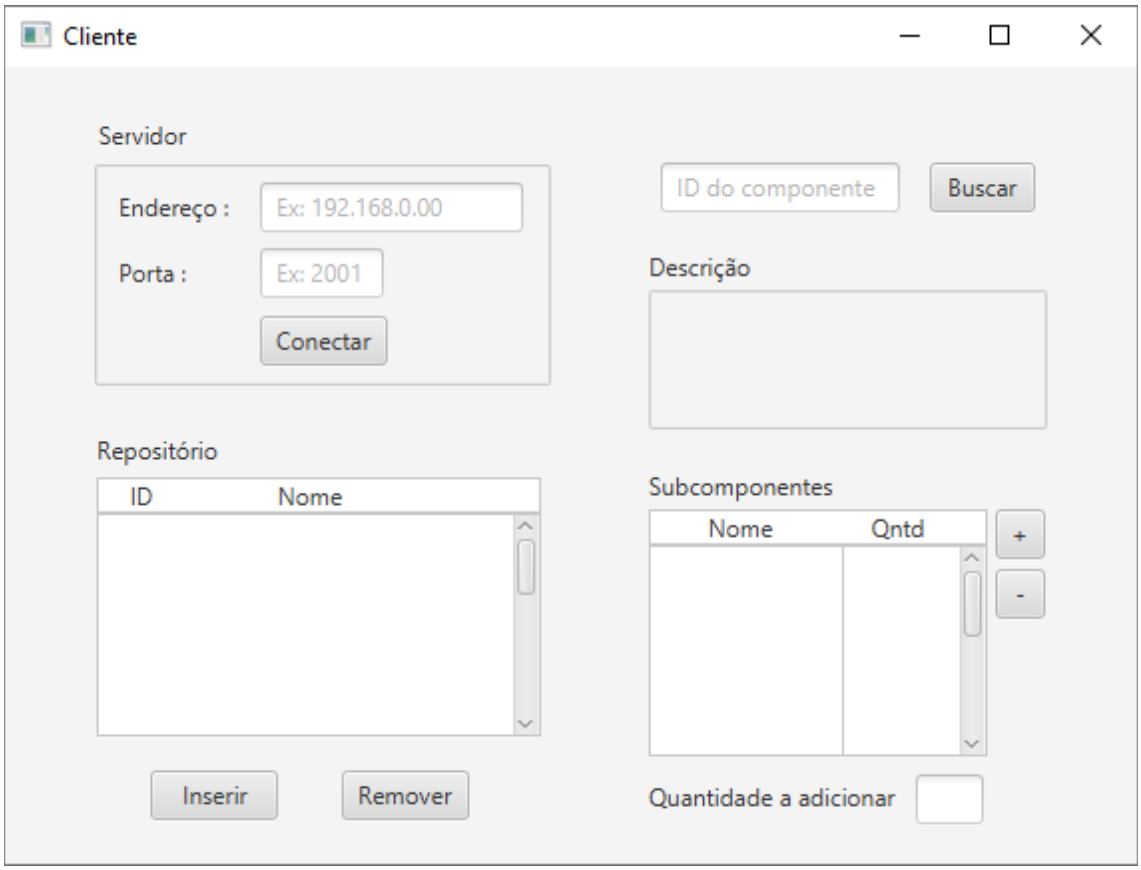

Figura 11: Tela de interação do Cliente com o Servidor.

GUIClientController.java é a classe controladora do arquivo fxml acima descrita. Esta classe é responsável por efetuar a conexão com o servidor (a mecânica de funcionamento está descrita no item 4. FUNCIONAMENTO) e compor as funções dos componentes de tela.

GUIInsertPart.fxml é o arquivo que contêm os componentes de tela da nova janela que surge ao usuário clicar no botão de inserir uma nova peça no repositório corrente. É necessário inserir o nome da peça e descrevê-la (o id é gerado automaticamente por `PartRepositoryConcrete.addPartToRepository()`).

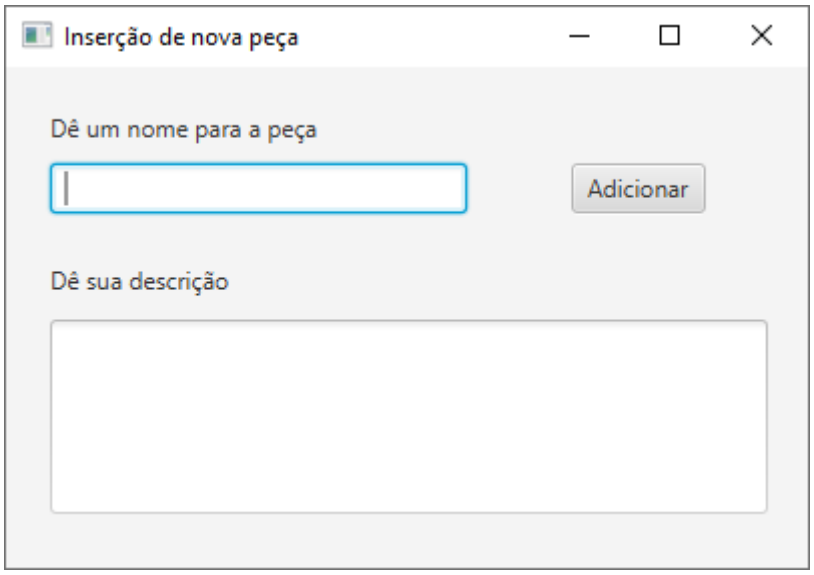

Figura 12: Inserção de nova peça ao repositório corrente.

GUIInsertPartController.java é a classe que atribui funcionalidades aos componentes de tela acima.

Main.java é a classe que possui o método start(), método chamado para a inicialização dos componentes de JavaFX, i.e., a tela inicial mostrada na Figura 11. Esta classe é inicializada na chamada da classe Client.Client.

PACKAGESERVER

Server.java é classe que possui os atributos do servidor como o ID, nome e repositório, e também suas funcionalidades.

RunServer.java é classe que cria instâncias de Servidor.classe, possui o método para que o cliente possa estabelecer conexão com uma das instâncias de servidor através do método `establishConnection()`, dado a instância Server e a porta de acesso como argumento.

REPOSITORIES

O repositório de cada servidor é definido em um arquivo txt, de forma que as operações de adição e remoção de peças são efetuadas de modo que estas peças são persistidas em arquivos txt no seguinte formato:

Id;Nome;Descrição

A classe que gerencia as operações no repositório, assim como a formatação das informações das peças, é o RepositoryHandler.java, que efetua as operações no txt (atualiza o txt com as peças adicionadas ou removidas pelo cliente), faz o tratamento dos arquivos, inicializa o repositório, dado o nome do repositório e o repositório em si (ArrayList) que é passado como argumento no método

```
RepositoryHandler.initializeRepositoryTxt()
```

SERVERAPIs

Part.java é a interface que possui os métodos a serem implementados. Esta interface é a entidade que define uma peça.

PartConcrete.java é a classe que implementa a interface Part.java. A partir desta classe é possível criar instâncias de peças para seu manuseio (adicionar ao repositório, remover, adicionar subcomponentes, consultar sua descrição, efetuar busca da mesma entre outros).

PartRepository.java é a interface que define o repositório que uma instância de servidor possuirá.

PartRepositoryConcrete.java é a classe que implementa PartRepository.java.

4. **FUNCIONAMENTO**

4.1 **LADO SERVIDOR**

A classe RunServer.java cria quatro instâncias de Server.java, define suas portas, cria um repositório para cada instância e estabelece uma conexão para cada uma das instâncias através do método `establishConnection()`, dado como argumento a instância e a porta definida.

O método cria um objeto da interface PartRepository vinculando-o com a sua respectiva porta através de:

```
UnicastRemoteObject.exportObject(objeto_repo.getRepository(), porta)
```

Posteriormente é criado um registro, dado a porta do servidor a ser conectado, para que o cliente possa se conectar (detalhes do estabelecimento por parte do cliente será descrito mais a frente). Por fim, o registro é vinculado ao repositório criado:

```
registry.bind(“ServerAPIs.PartRepository”, objPR);
```

Onde objPR é o objeto da interface PartRepository.

O programa servidor está ciente de todas as operações realizadas pelo lado do cliente. Na execução do servidor, lhe é aparecido a operação realizada pelo cliente, em algumas requisições o nome do método, a classe que este pertence, a quantidade solicitada, para as operações de adição de subcomponentes, por exemplo, e o nome do servidor a que o cliente solicita a requisição.

4.2 **LADO CLIENTE**

Agora do lado do cliente, ao definir uma porta válida (o endereço não é preenchido, já que simulação ocorre localmente) e clicar em conectar, a conexão é estabelecida através do método `GUIClientController.onConnectButtonCliecked()`, onde o registro do cliente é definido, assim como o repositório a ser buscado através do método de busca da instância da interface Registry de GUIClientController. Caso o usuário não insira porta alguma, a exceção é capturada, tratada e a janela de alerta do erro é mostrada ao usuário (Figura 1). Caso o usuário insira uma porta que não corresponda a um servidor existente, a exceção é capturada, tratada e é plotado a janela da Figura 2.

Na GUI do cliente, os componentes de tela foram organizados da seguinte forma: (a) Servidor, (b) Repositório, (c) Campo de busca, (d) Descrição e (e) Subcomponentes.

Então ao se conectar a uma porta válida, o usuário consegue visualizar, pelo campo (b), o nome do repositório corrente, a quantidade de peças e a lista de peças deste repositório onde são dados os IDs e os nomes de cada peça, como mostra a figura a seguir:

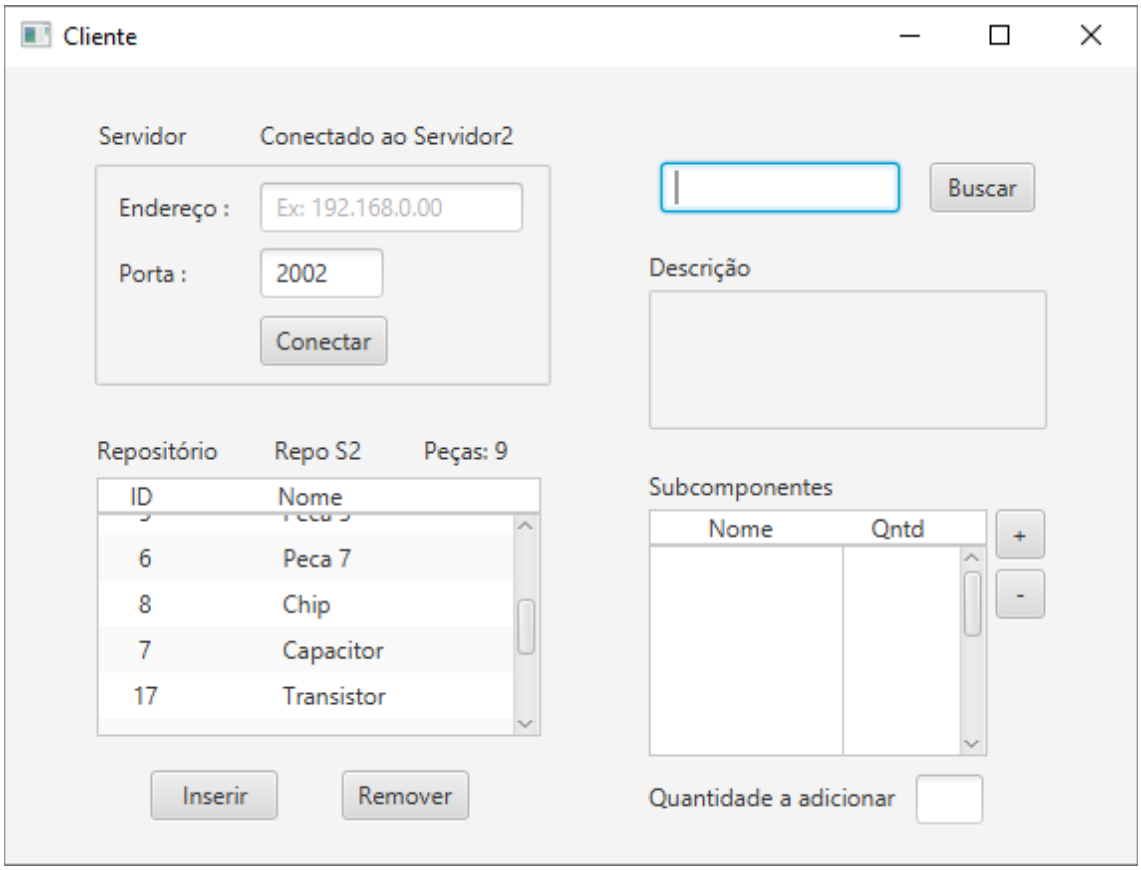

Figura 13: Cliente conectado ao servidor.

No campo (b), as especificações são mostradas através do método `showListViewServerRepository()` da classe GUIClientController.java, onde é definido o nome do repositório e a quantidade de peças. As informações são guardadas em uma matriz do tipo String. Esta matriz é inicializada com o retorno do método da interface `getRepoMatrix()` da classe PartRepositoryConcrete.java. Neste método uma matriz do tipo String (`repoMatrix[][]`) é criada e inicializada com a quantidade de linhas sendo igual a quantidade de peças existentes no repositório do servidor corrente (`int l = this.listPartRepository.size()`) e duas colunas, uma para o ID e outra para o nome da peça. E para cada peça (`Part part`) do array list, o ID que é um inteiro é convertido para String para que assim a matriz `repoMatrix[][]` possa receber o ID na primeira coluna e o nome da peça na segunda e ser passada como retorno do método. Assim, a matriz de String `toListView[][]` do método `showListViewServerRepository()` da classe GUIClientController.java recebe `repoMatrix[][]`. E posteriormente, impressa no componente list view da tela do Cliente com os IDs e nomes de cada uma das peças (como mostra a Figura 13).

Ao usuário clicar no botão “Inserir”, o método `onInsertButtonClicked()` é chamado. Este método inicia e carrega uma nova tela com a chamada do arquivo GUIInsertPart.fxml. E uma nova tela surge (Figura 12) com os campos para inserção do nome e descrição da peça a ser adicionado no repositório corrente e o botão “Adicionar”. As funcionalidades destes componentes são implementadas pela classe GUIInsertPartController.java. Quando o botão “Adicionar” é clicado, depois das validações de preenchimentos dos campos serem feitos, o método `addPartToRepository()` da classe PartRepositoryConcrete.java é chamado.

4.2.1. **GERAÇÃO DO ID**

O método `addPartToRepository()` recebe como argumento o nome e a descrição da peça. Esta classe a qual o método referenciado pertence, possui uma variável importante para a geração automática do ID da peça a ser adicionada no repositório, que é a variável `availableID` de tipo inteiro, inicializada com -1. Assim que uma peça é removida, `availableID` recebe o ID desta peça antes que a mesma seja removida, assim, o ID desta peça fica disponível para que quando uma nova peça seja adicionada, recebe o ID da antiga peça, que agora se encontra disponível para reúso.
Então ao adicionar uma nova peça, é verificado se `availableID` é diferente de -1, caso essa condição seja satisfeita, a nova peça é adicionada ao repositório com este ID. Recebendo este ID, `availableID` é reinicializada com -1. E no final do método `addPartToRepository()`, após as operações terem sido realizadas, `RepositoryHandler.addPartToRepoTxt()` é chamado, para que esta nova peça seja persistida no respecito txt do repositório corrente.
Para cada peça existente no ArrayList (que é o repositório do servidor), é feito uma verificação se o ID já existe, através da chamada do método `PartRepositoryConcrete.verifyIfIDAlreadyExist()`, se existir `availableID` é incrementado em uma unidade, até que o arraylist seja todo percorrido, e assim a nova peça é adicionada ao arraylist e um novo ID é criado.

O método `verifyIfIDAlreadyExist()` da classe PartRepositoryConcrete.java recebe como argumento ID da peça a ser manuseada. Este método tem a finalidade de verificar se o ID da peça solicitada existe, e caso exista, retorna o índice da posição desta peça no arraylist. Percorre-se cada peça do repositório e comparado se o argumento do método é igual ao ID da peça da lista, caso seja, a variável local ‘index’

recebe o índice desta peça no arraylist.

5. INSTRUÇÕES PARA A EXECUÇÃO DO CLIENTE-SERVIDOR

5.1 LINUX MINT 20.1 / UBUNTU 20.04 LTS

Para iniciar o servidor, abrir o terminal no diretório:
EP1/Versão para distros Linux (Ubuntu e Mint)/EP1/src
Inserir o seguinte comando:
> `java PackageServer/RunServer &`

Para executar o cliente, no caminho:
EP1/Versão para distros Linux (Ubuntu e Mint)/EP1/src
Versão para distros Linux (Ubuntu e Mint)/EP1/out/artifacts/EP1_jar
Dar dois cliques em EP1.jar

5.2 WINDOWS 10

Para iniciar o servidor, abrir o terminal no diretório:
EP1\Versão para Windows\EP1\src
e digitar o seguinte comando:
> `start java PackageServer/RunServer`

Para executar o cliente, no caminho
EP1\Versão para Windows\EP1\out\artifacts\EP_jar
Dar dois cliques em EP.jar