



**Universidade de São Paulo
Escola de Artes, Ciências e Humanidades**

Integrantes:

André Ramos	9125339
Matheus Mendes Sant'Ana	8921666
Victor Luiz Soares Alexandre Pereira	8921350

**1º Exercício Programa
Java™ Remote Method Invocation (RMI)**

**SÃO PAULO
2017**

Sumário

1) Introdução	3
1.1) Java™ Remote Method Invocation (Java RMI)	3
2) Pacotes	4
2.1) Pacote “Interfaces” – Definindo interfaces remotas	4
2.2) Pacote “Executar”	4
2.2.1) Implementando o Servidor – Server.java	4
2.2.2) Implementando o Cliente – Client.java	5
2.3) Pacote “Objetos”	5
2.4) Pacote “Util”	5
3) Interface do Sistema	6
4) Comandos	7
5) Tutorial de Inicialização	9
5.1) Windows	9
5.1.1) Eclipse	9
5.1.2) Linha de Comando (CMD)	11
5.1.3) Re-executar o sistema do início	12
5.2) Linux	13
5.2.1) Eclipse	13
5.2.2) Linha de Comando (CMD)	15
5.2.3) Re-executar o sistema do início	15
6) Exemplos de teste de execução	16
7) Referências Bibliográficas	27

1) Introdução

Para este 1º Execício Programa, descreveremos qual foi a solução implementada para resolver o problema de peças e componentes proposto, tanto do lado cliente como do lado servidor. Ilustraremos como será o acesso do sistema pela interface gráfica (estilo “linha de comandos”) através de um tutorial passo a passo, mencionando a maneira de fazer buscas, modificações e inserções de objetos – comandos tanto para entrada como para saída de dados.

Para a realização deste trabalho, contamos com o auxílio da biblioteca Java RMI (Remote Method Invocation): explicitaremos de forma clara os principais métodos, classes e variáveis utilizadas e suas respectivas funções dentro sistema. Seguiremos a documentação da biblioteca Java RMI de 2016, cujo link encontra-se na bibliografia deste documento.

1.1) Java™ Remote Method Invocation (Java RMI)

Java RMI é um mecanismo para permitir a invocação de métodos que residem em diferentes máquinas virtuais Java (JVM), ou seja, um mecanismo de chamada de procedimento remoto orientado a objetos.

2) Pacotes

Descreveremos os pacotes que constituem a arquitetura do sistema, bem como os arquivos presentes em cada um deles e daremos uma visão geral da importância das entidades pertencentes a cada um desses pacotes.

2.1) Pacote “Interfaces” – Definindo interfaces remotas

Um objeto remoto é uma instância de uma classe que implementa uma interface remota. Essas interfaces remotas herdam as propriedades da interface `java.rmi.Remote` e podem declarar um conjunto de métodos remotos. Cada método remoto deste tipo de interface inclui a exceção `java.rmi.RemoteException` na sua cláusula *throws*.

Os arquivos presentes neste pacote são: `Part.java`, `PartRepository.java` e `Subcomponente.java`. Cada um desses arquivos contém sua respectiva interface que será utilizada como uma referência a objetos remotos do sistema.

2.2) Pacote “Executar”

2.2.1) Implementando o Servidor – `Server.java`

O servidor possuirá um método *main* que criará uma instância do objeto remoto que receberá chamadas remotas e registrará esse mesmo objeto no Registro (*Registry*) Java RMI. O método estático `UnicastRemoteObject.exportObject` exporta o objeto para receber invocações numa porta TCP e retorna uma referência (*stub*) para o objeto remoto passar pelos clientes.

Para os clientes conseguirem acesso a um objeto remoto, eles necessitarão primeiro obter uma referência (*stub*) do objeto remoto. Assim que um objeto remoto for registrado no servidor, os clientes conseguem localizá-lo e invocar métodos dele.

O método estático `LocateRegistry.getRegistry` retorna um *stub* e manda invocações no Registro do servidor (no exemplo do EP utiliza-se a porta 1099). O método *bind* é invocado para ligar as referências do objeto remoto à classe `PartRepository`.

A classe `Server` implementa a interface “`PartRepository`”. Foi definida a quantidade de 3 repositórios no sistema (para fins de teste), quantidade essa que pode ser alterada à maneira do programador, levando em conta também a quantidade de servidores no sistema e as diferentes portas disponíveis (tendo em vista uma aplicação real). Os nomes dos repositórios – a priori – que podem acessados são “**Repo1**”, “**Repo2**” e “**Repo3**”.

2.2.2) Implementando o Cliente – Client.java

O programa do lado “cliente” também possui um método *main* – assim como o programa do lado “servidor” – e obtém a referência (*stub*) no servidor; olha a referência do objeto remoto pelo nome no registro e pode invocar os métodos de PartRepository, buscando ou inserindo uma peça, por exemplo.

2.3) Pacote “Objetos”

Implementando as classes dos principais objetos – Componente.java e PartExample.java.

A classe **Componente** herda as propriedades da classe UnicastRemoteObject e implementa a interface Subcomponente (remota), além de instanciar os seguintes atributos: **peca** do tipo “Part” e **quantidade** do tipo “int”. Os métodos e o construtor da classe Componente tratam a exceção java.rmi.RemoteException em suas respectivas cláusulas *throws*, retornando ou imputando valor aos atributos da classe.

A classe **PartExample** herda as propriedades da classe UnicastRemoteObject e implementa a interface Part, além de instanciar os seguintes atributos: **codigo** do tipo “int”, **nome** do tipo “String”, **descricao** do tipo “String” e **subcomponentes** do tipo “LinkedList<Subcomponente>”. Os métodos e o construtor da classe tratam a exceção java.rmi.RemoteException em suas respectivas cláusulas *throws*.

2.4) Pacote “Util”

Este pacote possui apenas o arquivo Sistema.java, proprietário da classe Sistema, a qual contém os métodos responsáveis por verificar a existência de uma peça em um determinado repositório, verificar a existência de uma peça em todo o sistema, e também realizar a busca de uma peça que encontra-se em outro repositório que não seja o atual.

3) Interface do Sistema

O *console* do sistema tem a seguinte configuração:

repositórioAtual\ Peca:códigoDaPeçaAtual\ Sublista:códigoDaPeçaNova>

No começo da execução do programa (do lado do cliente), o console estará apresentado do seguinte modo:

Repo1\ Peca:\ Sublista:>

Como é possível perceber, o sistema tem como estado inicial o cliente conectado ao repositório de nome “**Repo1**”; sem nenhuma peça atual (peça “corrente”) – ou seja, nenhum código após o campo “**Peça:**” – e sem nenhuma peça pendente para ser inserida – nenhum código após o campo “**Sublista:**”.

Os comandos do sistema serão aprofundados na seção 4, todavia, para esclarecimento da configuração do *console*, apresentaremos três exemplos a seguir de modificações no sistema que o alteram:

1-) Suponha que o sistema acabou de ser iniciado e tinha a configuração inicial do console na forma **Repo1\ Peca:\ Sublista:>**. Suponha que executamos o comando **bind Repo2**. Depois do comando, o console apresentará a configuração **Repo2\ Peca:\ Sublista:>** pois o repositório atual foi alterado para “Repo2”.

2-) Suponha que o sistema acabou de ser iniciado e tinha a configuração inicial do console na forma **Repo1\ Peca:\ Sublista:>**. Suponha que temos a peça de código **1** no repositório atual e executamos o comando **getp 1**. Depois do comando, o console apresentará a configuração **Repo1\ Peca:1\ Sublista:>** pois a peça atual foi alterada para a peça de código “1”.

3-) Suponha que o sistema acabou de ser iniciado e tinha a configuração inicial do console na forma **Repo1\ Peca:\ Sublista:>**. Suponha que não há qualquer peça no sistema e executamos o comando **addp**, preenchendo após este comando – de acordo com o que a interface solicita – o código da peça que será **1**, o nome que será “**peca1**” e descrição que será “**peca1**”. Depois do comando, o console apresentará a configuração **Repo1\ Peca:\ Sublista:1>** pois a peça atual que está sendo criada foi alterada para a peça de código “1”.

4) Comandos

bind: Troca o repositório com o qual o cliente está se comunicando. Este comando deve ser seguido pelo novo repositório com o qual o cliente quer se comunicar, exemplo: bind Repo2. O nome do repositório que vem logo após o comando bind pode ser “**Repo1**”, “**Repo2**” ou “**Repo3**” (sem as aspas), lembrando que esses nomes podem estar em caixa alta ou baixa – utilizar “repo1” ao invés de “Repo1” também é uma maneira válida de se comunicar com o repositório “Repo1”.

addp: Ao executar este comando para inserir uma peça, será requisitado em um primeiro momento o código da peça que se pretende inserir. Ao inserir o código da peça, existem 4 cenários possíveis para o programa:

1º) O código não é numérico: o programa pedirá novamente uma entrada numérica (deve ser um número inteiro) para o código da peça.

2º) Já existe uma peça com o código inserido no repositório atual: o programa voltará ao estado anterior antes de ser requisitado o comando “addp”.

3º) Já existe uma peça com o código inserido em um outro repositório: o repositório atual receberá uma referência da peça encontrada com este código.

4º) Não existe uma peça no sistema que esteja cadastrada com o código inserido: serão requisitados o nome e a descrição da peça. Após inserir o nome e a descrição, a lista de subpeças atual passa a ser a lista de subpeças desta nova peça que se pretende adicionar ao repositório.

Durante a criação de uma peça pode-se executar o comando “**cancel**” para cancelar esta operação de inserção. Após o preenchimento do código, nome e descrição de uma peça – isto é, após a consolidação do 4º passo – (conforme a interface pede), é possível concluir a criação da peça através do comando “**terminate**”, quer a lista de subpeças tenha sido preenchida ou não (conforme o enunciado do EP, a peça pode ser primitiva ou agregada). Após a confirmação da adição de uma peça a um repositório (comando “**terminate**”), não é possível modificá-la mais.

terminate: Mencionado na descrição do comando “addp”, este comando conclui a inserção de uma peça no repositório atual, quer a lista de subpeças tenha sido preenchida com outras peças ou não.

cancel: Mencionado na descrição do comando “addp”, este comando cancela a inserção de uma peça no repositório atual.

clearlist: (Funciona apenas durante a criação de uma peça. Utilize este comando antes do comando “**terminate**”): Remove todas as subpeças da lista de subpeças atual.

getp: Busca por uma peça no repositório atual. O comando getp deve ser seguido pelo código da peça procurada, exemplo: getp 1. Se existe uma peça com o código procurado no repositório atual, ela passa a se tornar a nova peça atual. Do contrário – se a busca retornou “null” – a peça atual continua sendo a peça anterior à busca.

getsubpart: Busca por um subcomponente na peça atual. O comando getsubpart deve ser seguido pelo código do subcomponente procurado pertencente à peça atual, exemplo: getsubpart 1. Se existe um subcomponente com o código procurado na peça atual, este subcomponente passa a se tornar a nova peça atual. Do contrário – se a busca retornou “null” – a peça atual continua sendo a peça anterior à busca.

addsubpart: (Apenas durante a criação de uma peça. Utilize este comando antes do comando “terminate”): Adiciona **n** unidades da peça atual à peça que está sendo criada. Caso a peça atual seja “null” ou não esteja sendo criada uma nova peça, é retornada uma mensagem de erro. O comando deve ser seguido pela quantidade **n**, por exemplo: addsubpart 77. Descrevendo a operação em “baixo nível”, temos que a peça que está sendo criada tem uma lista ligada de objetos do tipo “Componente”; para adicionarmos **n** unidades de uma peça de código **k** à lista, podem ocorrer 2 cenários:

1º) A lista de componentes já possui um componente de código k: neste caso o nó “Componente” que contém a peça de código **k** será modificado, recebendo a quantidade atual somada à quantidade anterior.

2º) A lista de componentes ainda não possui um componente de código k: neste caso será criado e inserido na lista um novo objeto “Componente” com a peça de código **k** e a quantidade **n**.

showp: Se a peça atual não for “null”, o comando mostra o código, nome e descrição da peça, além de exibir se a peça é agregada ou primitiva. Caso a peça atual seja agregada, são mostrados os atributos dos subcomponentes que ela contém, sendo eles código, nome, descrição e quantidade de cada uma das peças que a compõe.

listp: Mostra a quantidade de peças que existem no repositório atual e lista o código, nome e descrição de cada uma dessas peças presentes no repositório atual.

listrepos: Lista os nomes dos repositórios cadastrados atualmente no registro do Java RMI.

quit: Termina a execução do programa do lado “cliente”.

5) Tutorial de Inicialização

5.1) Windows

As seções 5.1.1 e 5.1.2 são duas formas de configurar o ambiente de execução do sistema, sendo que pode-se utilizar uma ou outra (não necessariamente as duas).

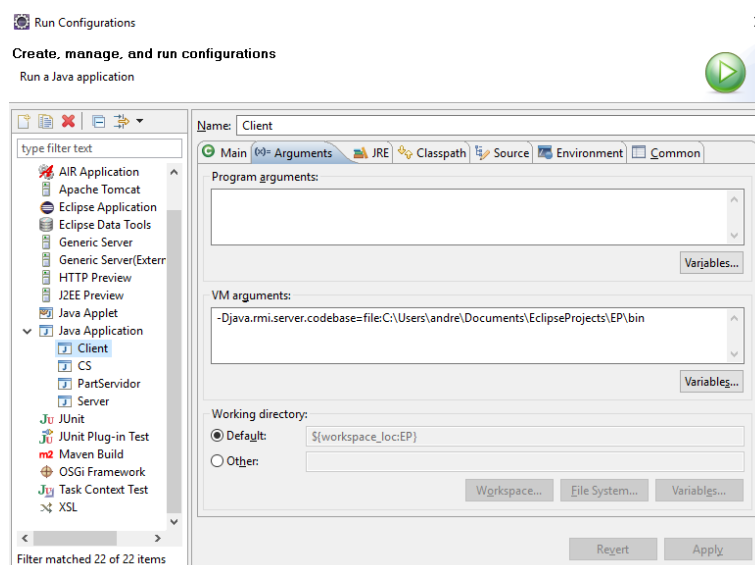
5.1.1) Eclipse

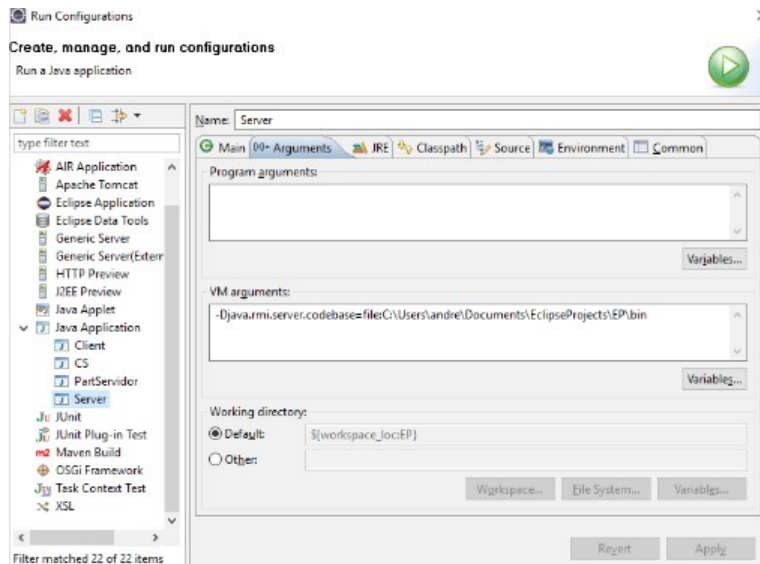
1-) Crie um projeto no Eclipse (ATENÇÃO: seu *workspace* NÃO pode conter espaço no nome das pastas Ex: ...\\Eclipse Projects\\.... , caso tenha espaço, recrie o *workspace* do Eclipse na inicialização).

2-) Ao criar o novo projeto, certifique-se também de que o nome NÃO contenha espaços. Na pasta “src” do projeto do Eclipse adicione (copie e cole) os *packages* do EP (Objetos, Executar, Util e Interfaces). Em seguida, dê um *refresh* na pasta src (clcando com o botão direito em cima) para que apareça no Eclipse os *packages* adicionados.

3-) Ainda com o projeto aberto no Eclipse, seguir o caminho Run → Run Configurations → Arguments. No campo VM Arguments inserir o seguinte comando no filtro das classes Client e Server de Java Application: “-Djava.rmi.server.codebase=file:\${workspace_loc}\\\${project_name}\\bin” (sem as aspas) e clicar em Apply. O pedaço `${workspace_loc}` do código é seu *workspace* e o pedaço `${project_name}` é o nome do projeto (esses pedaços podem ser substituídos pelos nomes verdadeiros, como é mostrado abaixo, mas não é necessário).

Exemplo: “-Djava.rmi.server.codebase=file:C:\\Users\\andre\\Documents\\EclipseProjects\\EP\\bin”

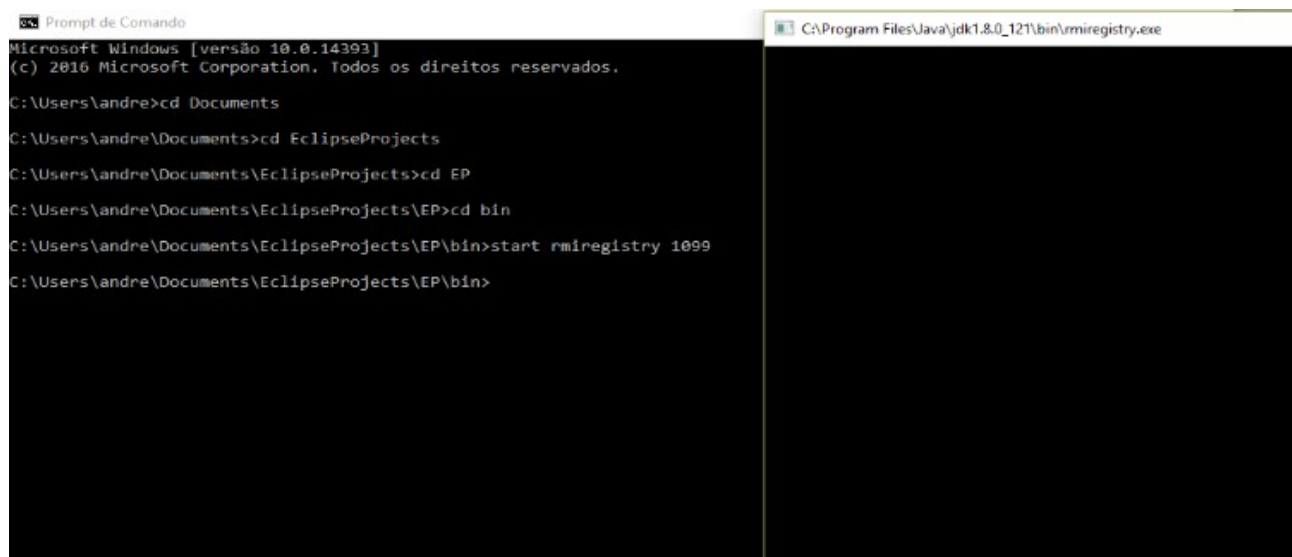




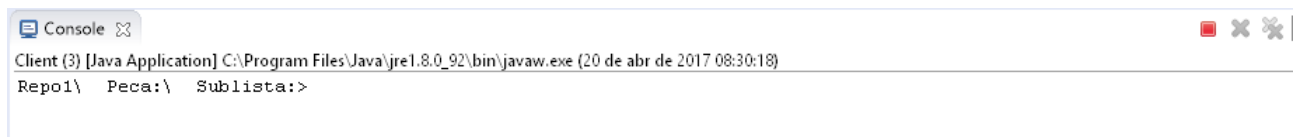
4-) Abrir o terminal e mudar o diretório para a pasta “bin” do projeto do Eclipse (diretório raiz onde estão localizados os arquivos .class do projeto). Novamente, verifique se todas as pastas do caminho do diretório NÃO contém espaço no nome, do contrário mude o *workspace* do Eclipse para um diretório que não tenha esse problema.

5-) Executar no terminal o seguinte comando: `start rmiregistry 1099`

Exemplo:



6-) Esperar em torno de 5 segundos. Executar no Eclipse a classe Server, esperar até obter a mensagem “Servers ready” e executar a classe Client em seguida. O console deverá apresentar a seguinte configuração:



```
Client (3) [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (20 de abr de 2017 08:30:18)
Repo1\ Peca:\ Sublista:>
```

Após ter concluído todas as etapas corretamente, o sistema estará pronto para receber os comandos.

5.1.2) Linha de Comando (CMD)

1-) Escolher um diretório raiz que conterá todos os arquivos do projeto (os arquivos **.java** e **.class**); utilizaremos como exemplo **C:\Users\Usuario\Desktop\EP** – a pasta **Usuario** deve ter o nome do seu computador. Esse diretório deve possuir permissão do explorador de arquivos do computador, do contrário poderá ocorrer erro(s).

2-) Copiar os 4 *packages* do EP para o diretório escolhido no passo 1.

3-) Abrir o terminal e mudar o diretório para o diretório escolhido no passo 1.

4-) No mesmo terminal do passo 3 compilar os **.java** através do seguinte comando:

```
javac Objetos/*.java Executar/*.java Interfaces/*.java Util/*.java
```

5-) No mesmo terminal do passo 3 executar os seguintes comandos:

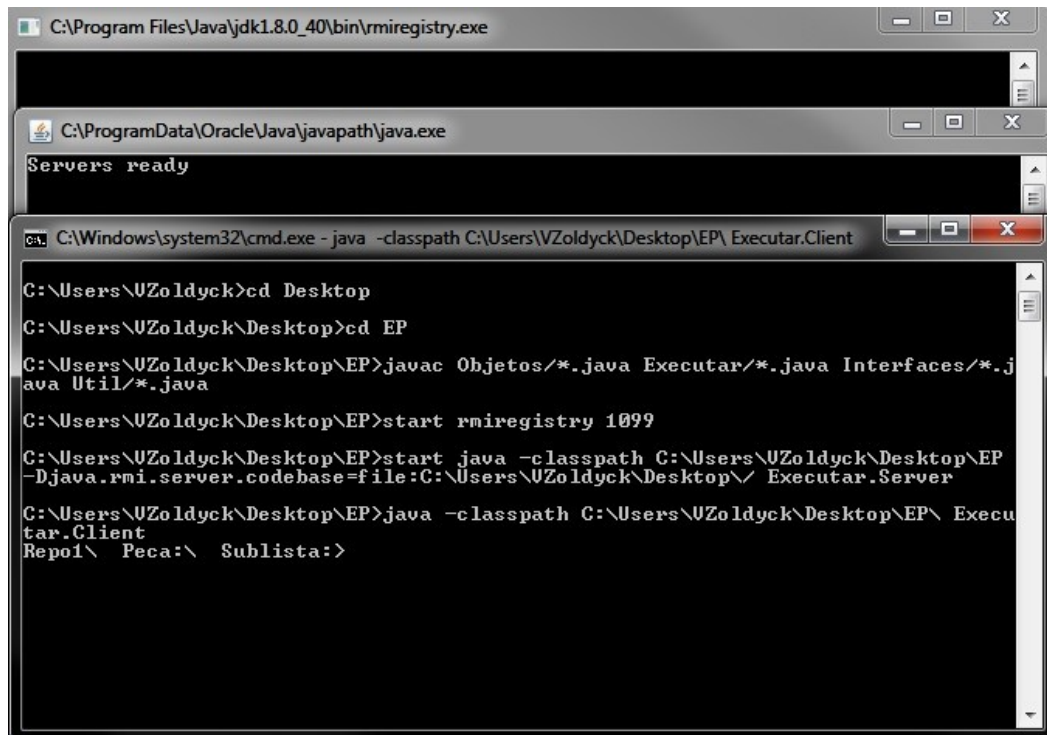
```
start rmiregistry 1099
```

```
start java -classpath C:\Users\Usuario\Desktop\EP\ -Djava.rmi.server.codebase=file:C:\Users\Usuario\Desktop\EP\
Executar.Server
```

```
java -classpath C:\Users\Usuario\Desktop\EP\ Executar.Client
```

Obs: Note que o segundo e o terceiro comando acima utilizam **Executar.Server** e **Executar.Client**, o que pode ser modificado para implementações diferentes deste EP. **Executar** é o *package* em que as classes **Server** e **Client** estão inseridas. Se houvesse outro *package* anterior a **Executar**, como por exemplo **Iniciar**, executaríamos os comandos **Iniciar.Executar.Server** e **Iniciar.Executar.Client** ao invés dos comandos anteriores.

Para ilustrar os passos descritos acima, segue uma imagem que mostra a sequência de passos que devem ser executados para completar a inicialização pelo terminal.



```
C:\Program Files\Java\jdk1.8.0_40\bin\rmiregistry.exe
C:\ProgramData\Oracle\Java\javapath\java.exe
Servers ready
C:\Windows\system32\cmd.exe - java -classpath C:\Users\UZoldyck\Desktop\EP\ Executar.Client
C:\Users\UZoldyck\Desktop>cd Desktop
C:\Users\UZoldyck\Desktop>cd EP
C:\Users\UZoldyck\Desktop\EP>javac Objetos/*.java Executar/*.java Interfaces/*.java Util/*.java
C:\Users\UZoldyck\Desktop\EP>start rmiregistry 1099
C:\Users\UZoldyck\Desktop\EP>start java -classpath C:\Users\UZoldyck\Desktop\EP -Djava.rmi.server.codebase=file:C:\Users\UZoldyck\Desktop\EP\ Executar.Server
C:\Users\UZoldyck\Desktop\EP>java -classpath C:\Users\UZoldyck\Desktop\EP\ Executar.Client
Repo1\ Peca:\ Sublista:>
```

5.1.3) Re-executar o sistema do início

Para executar a aplicação do zero novamente – isto é, executar o sistema após limpar completamente os dados da aplicação, tanto dos repositórios como do lado “servidor” e do lado “cliente” –, **fechar a janela de execução do servidor** (ou simplesmente parar a execução do servidor, seja através do comando “stop” no Eclipse ou através do comando ^C no terminal), **fechar a janela de execução do registro RMI** (*rmiregistry*) e **parar a execução da classe Client** (seja através do comando “stop” no Eclipse ou através do comando ^C no terminal). Após seguidos os 3 passos, seguir o tutorial de inicialização (seção 5) novamente.

5.2) Linux

As seções 5.2.1 e 5.2.2 são duas formas de configurar o ambiente de execução do sistema, sendo que pode-se utilizar uma ou outra (não necessariamente as duas).

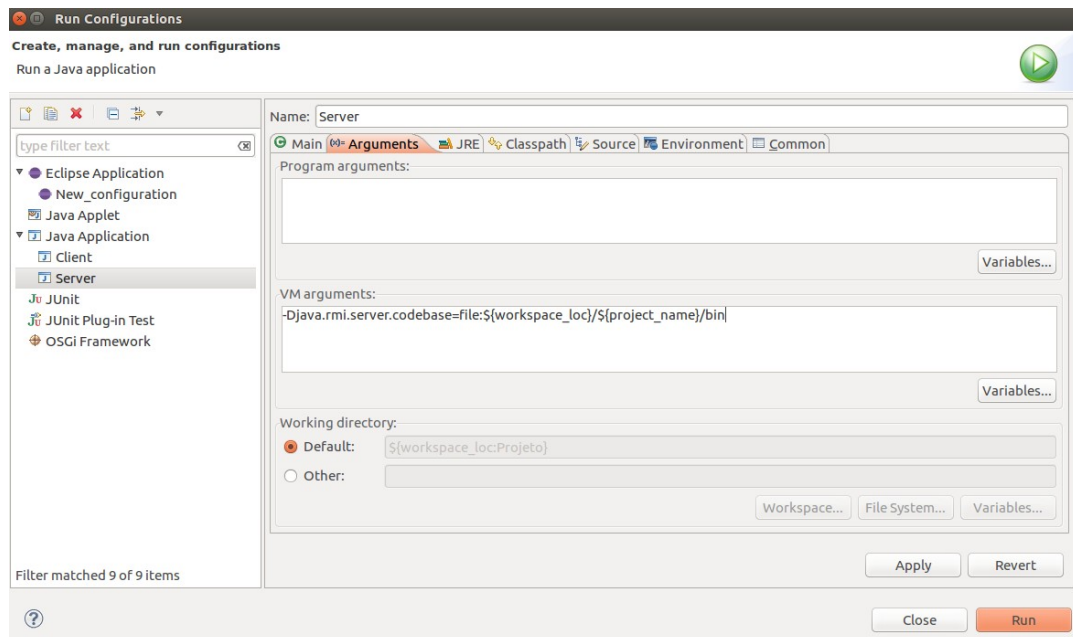
5.2.1) Eclipse

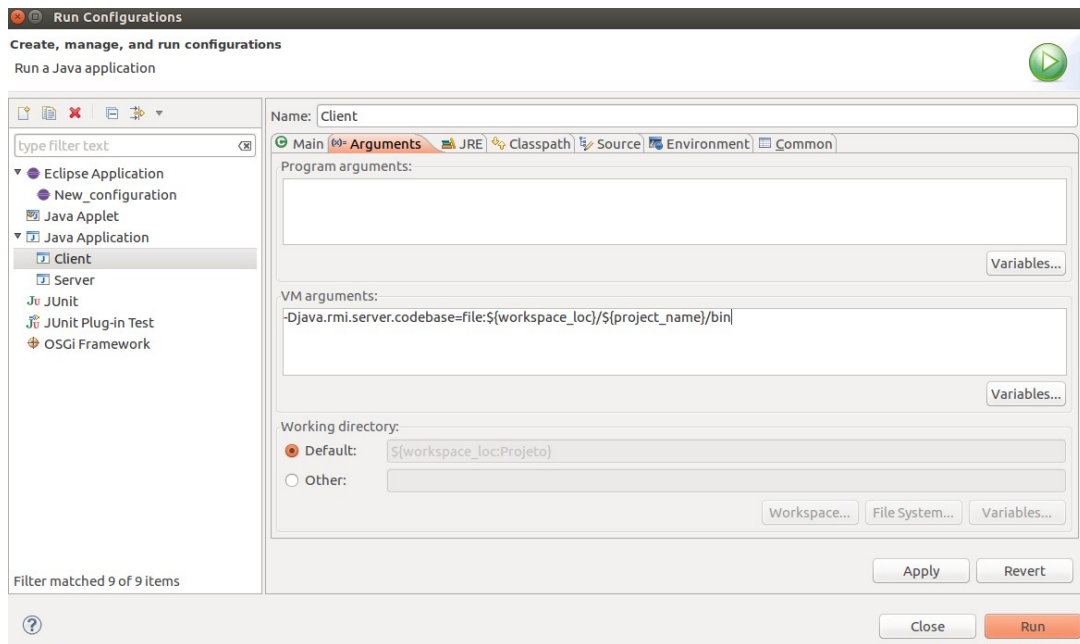
1-) Crie um projeto no Eclipse (ATENÇÃO : seu *workspace* NÃO pode conter espaço no nome das pastas Ex: .../Eclipse Projects/.... , caso tenha espaço, recrie o *workspace* do Eclipse na inicialização).

2-) Ao criar o novo projeto, certifique-se também de que o nome do projeto NÃO contenha espaços. Na pasta “src” do projeto do Eclipse adicione (copie e cole) os *packages* do EP (Objetos, Executar, Util e Interfaces). Em seguida, dê um *refresh* na pasta src (clitando com o botão direito em cima) para que apareça no Eclipse os *packages* adicionados.

3-) Ainda com o projeto aberto no Eclipse, seguir o caminho Run → Run Configurations → Arguments. No campo VM Arguments inserir o seguinte comando no filtro das classes Client e Server de Java Application:

“-Djava.rmi.server.codebase=file:\${workspace_loc}/\$ {project_name}/bin” (sem as aspas) e clicar em Apply. O pedaço `$ {workspace_loc}` do código é seu *workspace* e o pedaço `$ {project_name}` é o nome do projeto (esses pedaços podem ser substituídos pelos nomes verdadeiros, mas não é necessário).

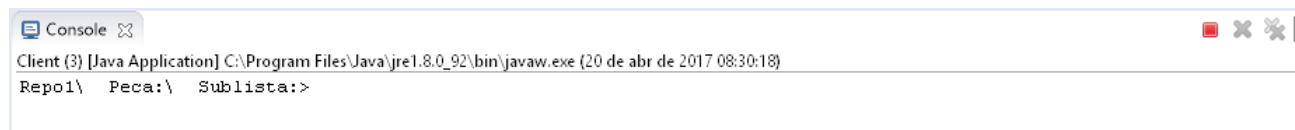




4-) Abrir o terminal e mudar o diretório para a pasta “bin” do projeto do Eclipse (diretório raiz onde estão localizados os arquivos .class do projeto). Novamente, verifique se todas as pastas do caminho do diretório NÃO contêm espaço no nome, do contrário mude o *workspace* do Eclipse para um diretório que não tenha esse problema.

5-) Executar no terminal o seguinte comando: `rmiregistry 1099 &` (caso haja erro rodar apenas `rmiregistry &`)

6-) Esperar em torno de 5 segundos. Executar no Eclipse a classe `Server`, esperar até obter a mensagem “Servers ready” e executar a classe `Client` em seguida. O console deverá apresentar a seguinte configuração:



Após ter concluído todas as etapas corretamente, o sistema estará pronto para receber os comandos.

5.2.2) Linha de comando (CMD)

1-) Escolher um diretório raiz que conterá todos os arquivos do projeto (os arquivos **.java** e **.class**); utilizaremos como exemplo **/root/Documentos/Project**. Esse diretório deve possuir permissão do explorador de arquivos do computador, do contrário poderá ocorrer erro(s).

2-) Copiar os 4 *packages* do EP para o diretório escolhido no passo 1.

3-) Abrir o terminal e mudar o diretório para o diretório escolhido no passo 1.

4-) No mesmo terminal do passo 3 compilar os **.java** através do seguinte comando:

```
javac Objetos/*.java Executar/*.java Interfaces/*.java Util/*.java
```

5-) No mesmo terminal do passo 3 executar os seguintes comandos:

```
rmiregistry 1099 &
```

```
java -classpath /root/Documentos/Project -Djava.rmi.server.codebase=file:/root/Documentos/Project Executar.Server &
```

6-) Manter o terminal do Servidor (do passo 3) executando e abrir um novo terminal no mesmo diretório que o passo 1. Rodar neste novo terminal o seguinte comando:

```
java -classpath /root/Documentos/Project Executar.Client
```

5.2.3) Re-executar o sistema do início

Para executar a aplicação do zero novamente – isto é, executar o sistema após limpar completamente os dados da aplicação, tanto dos repositórios como do lado “servidor” e do lado “cliente” –, **parar a execução do servidor** (seja através do comando “stop” no Eclipse ou através do comando ^C no terminal), **parar a execução do registro RMI** (*rmiregistry*) – se o registro Java estiver executando é possível localizá-lo pelo comando % fg no terminal e executar o comando ^C para finalizá-lo –, e **parar a execução da classe Client** (seja através do comando “stop” no Eclipse ou através do comando ^C no terminal). Após seguidos os 3 passos, seguir o tutorial de inicialização (seção 5) novamente.

6) Exemplos de teste de execução

1 – Utilizando todos os comandos:

Popularemos nosso repositório e utilizaremos os comandos desenvolvidos para uma melhor compreensão do funcionamento do sistema. Para explorar as possibilidades oferecidas, basta seguir o tutorial de inicialização (seção 5) e seguir os comandos abaixo:

listrepos (listando repositórios cadastrados no registro Java RMI)

bind repo1 (acesso ao repositório 1)

listp (listar as peças contidas no repositório atual, atualmente nenhuma peça)

addp (adicionar peça, ao executar o comando serão requisitados um código, nome e descrição; como sugestão digite na ordem: 1, p1, p1p1)

terminate (conclui a inserção da peça de código 1 no repositório 1)

listp (listar as peças contidas no repositório atual, atualmente apenas a peça de código 1)

addp (adicionar peça)

cancel (cancela a inserção)

bind repo2 (acesso ao repositório 2)

addp (adicionar peça, ao executar o comando serão requisitados o código, nome e descrição; como sugestão digite na ordem: 2, p2, p2p2)

bind repo1 (muda para o repositório onde está a peça 1)

getp 1 (recupera a peça 1)

addsubpart 100 (adicionar na peça de código 2 a quantidade de 100 peças de código 1)

clearlist (limpa a lista de subcomponentes da peça 2)

addsubpart 70 (adicionar na peça de código 2 a quantidade de 70 peças de código 1)

terminate (conclui a inserção da peça 2 no repositório 2)

bind repo2 (muda para o repositório onde a peça 2 foi inserida)

listp (lista as peças existentes no repositório 2, atualmente apenas a peça 2)

getp 2 (recupera a peça 2)

showp (mostra a peça 2)

getsubpart 1 (recupera o subcomponente de código 1)

showp (mostra os atributos do subcomponente 1)

quit (finaliza a execução do cliente)

```
Repo1\ Peca:\ Sublista:> listrepos
Repo1
Repo2
Repo3

Repo1\ Peca:\ Sublista:> bind repo1
Repo1\ Peca:\ Sublista:> listp
Total de pecas: 0

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 1
Repo1\ Peca:\ Sublista:> Digite o nome da peca: p1
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: p1p1
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:1> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: p1 Descricao: p1p1
Total de pecas: 1

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: cancel
Insercao cancelada

Repo1\ Peca:\ Sublista:> bind repo2
Repo2\ Peca:\ Sublista:> addp
Repo2\ Peca:\ Sublista:> Digite o codigo da peca: 2
Repo2\ Peca:\ Sublista:> Digite o nome da peca: p2
Repo2\ Peca:\ Sublista:> Digite a descricao da peca: p2p2
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo2\ Peca:\ Sublista:2> bind repo1
Repo1\ Peca:\ Sublista:2> getp 1
Peca recuperada!

Repo1\ Peca:1\ Sublista:2> addsubpart 100
Peca(s) adicionada(s)!

Repo1\ Peca:1\ Sublista:2> clearlist
Lista limpa!
```

Continuação:

```
Repo1\ Peca:1\ Sublista:2> addsubpart 70
Peca(s) adicionada(s)!

Repo1\ Peca:1\ Sublista:2> terminate
Insercao concluida!

Repo1\ Peca:1\ Sublista:> bind repo2
Repo2\ Peca:1\ Sublista:> listp
Codigo: 2 Nome: p2 Descricao: p2p2
Total de pecas: 1

Repo2\ Peca:1\ Sublista:> getp 2
Peca recuperada!

Repo2\ Peca:2\ Sublista:> showp
Codigo: 2 Nome: p2 Descricao:p2p2
Peca agregada, subcomponentes:
    Codigo: 1 Nome: p1 Descricao: p1p1 [Quantidade = 70]

Repo2\ Peca:2\ Sublista:> getsubpart 1
Peca recuperada!

Repo2\ Peca:1\ Sublista:> showp
Codigo: 1 Nome: p1 Descricao:p1p1
Peca primitiva

Repo2\ Peca:1\ Sublista:> quit
Bye
```

2 – Tentar acessar um repositório inexistente:

```
C:\Users\andre\Desktop>java -classpath C:\Users\andre\Desktop\ Executar.Client
Picked up JAVA_TOOL_OPTIONS: -Djava.vendor="Sun Microsystems Inc."
Repo1\ Peca:\ Sublista:> bind repo3231
Repositorio inexistente

Repo1\ Peca:\ Sublista:> bind rep323
Repositorio inexistente

Repo1\ Peca:\ Sublista:> bind re100
Repositorio inexistente

Repo1\ Peca:\ Sublista:> bind 100
Repositorio inexistente

Repo1\ Peca:\ Sublista:> bind
Repositorio inexistente

Repo1\ Peca:\ Sublista:> bind1000
Repositorio inexistente

Repo1\ Peca:\ Sublista:> bind -3214
Repositorio inexistente
```

3 – Inserindo e recuperando uma peça com código negativo:

```
Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: -32
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca1
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: teste com numero negativo
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:-32> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> getp -32
Peca recuperada!

Repo1\ Peca:-32\ Sublista:> showp
Codigo: -32 Nome: peca1 Descricao: teste com numero negativo
Peca primitiva

Repo1\ Peca:-32\ Sublista:> listp
Codigo: -32 Nome: peca1 Descricao: teste com numero negativo
Total de pecas: 1
```

4 – Inserindo subpartes e verificando a sintaxe do comando:

```
Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 1
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca1
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: peca1
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:1> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> getp 1
Peca recuperada!

Repo1\ Peca:1\ Sublista:> addp
Repo1\ Peca:1\ Sublista:> Digite o codigo da peca: 123
Repo1\ Peca:1\ Sublista:> Digite o nome da peca: 123
Repo1\ Peca:1\ Sublista:> Digite a descricao da peca: peca para teste
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:1\ Sublista:123> addsubpart
Digite uma quantidade apos o comando 'addsubpart'

Repo1\ Peca:1\ Sublista:123> addsubpart quantidade
Digite apenas um numero inteiro apos o comando 'addsubpart'

Repo1\ Peca:1\ Sublista:123> addsubpart 0
Insira apenas valores positivos para a quantidade de subcomponentes

Repo1\ Peca:1\ Sublista:123> addsubpart -150
Insira apenas valores positivos para a quantidade de subcomponentes

Repo1\ Peca:1\ Sublista:123> addsubpart 999
Peca(s) adicionada(s)!
```

5 – Tentando adicionar uma nova peça durante a inserção de uma peça.

```
Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 000
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca nova
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: peca 000 nova
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:0> addp
Insercao da peca de codigo 0 cancelada.

Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 000
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca nova
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: peca 000 nova
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:0> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> listp
Codigo: 0 Nome: peca nova Descricao: peca 000 nova
Total de pecas: 1
```

6 – Fazendo cópia de uma peça existente em outro repositório:

```
Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 5
Repo1\ Peca:\ Sublista:> Digite o nome da peca: p5
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: p5p5
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:5> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> listp
Codigo: 5 Nome: p5 Descricao: p5p5
Total de pecas: 1

Repo1\ Peca:\ Sublista:> bind repo2
Repo2\ Peca:\ Sublista:> addp
Repo2\ Peca:\ Sublista:> Digite o codigo da peca: 5
Peca inserida em outro servidor... copia concluida!

Repo2\ Peca:\ Sublista:> listp
Codigo: 5 Nome: p5 Descricao: p5p5
Total de pecas: 1

Repo2\ Peca:\ Sublista:> bind repo3
Repo3\ Peca:\ Sublista:> addp
Repo3\ Peca:\ Sublista:> Digite o codigo da peca: 5
Peca inserida em outro servidor... copia concluida!

Repo3\ Peca:\ Sublista:> listp
Codigo: 5 Nome: p5 Descricao: p5p5
Total de pecas: 1
```

7 – Tentando inserir uma peça já existente no repositório atual:

```
Repo1\ Peca:\ Sublista:> listp
Total de pecas: 0

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 10
Repo1\ Peca:\ Sublista:> Digite o nome da peca: p10
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: p10p10
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:10> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> listp
Codigo: 10 Nome: p10 Descricao: p10p10
Total de pecas: 1

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 10
Esta peca ja foi inserida no repositorio atual

Repo1\ Peca:\ Sublista:>
```

8 – Inserindo uma peça primitiva e outra agregada:

```
Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 1
Repo1\ Peca:\ Sublista:> Digite o nome da peca: p1
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: p1p1
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:1> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 2
Repo1\ Peca:\ Sublista:> Digite o nome da peca: p2
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: p2p2
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:2> getp 1
Peca recuperada!

Repo1\ Peca:1\ Sublista:2> addsubpart 20
Peca(s) adicionada(s)!

Repo1\ Peca:1\ Sublista:2> terminate
Insercao concluida!

Repo1\ Peca:1\ Sublista:> listp
Codigo: 1 Nome: p1 Descricao: p1p1
Codigo: 2 Nome: p2 Descricao: p2p2
Total de pecas: 2

Repo1\ Peca:1\ Sublista:> getp 1
Peca recuperada!

Repo1\ Peca:1\ Sublista:> showp
Codigo: 1 Nome: p1 Descricao:p1p1
Peca primitiva

Repo1\ Peca:1\ Sublista:> getp 2
Peca recuperada!

Repo1\ Peca:2\ Sublista:> showp
Codigo: 2 Nome: p2 Descricao:p2p2
Peca agregada, subcomponentes:
Codigo: 1 Nome: p1 Descricao: p1p1 [Quantidade = 20]
```

9 – Limpando a lista de subpeças:

```
Repo1\ Peca:\ Sublista:> bind repo2
Repo2\ Peca:\ Sublista:> listp
Total de pecas: 0

Repo2\ Peca:\ Sublista:> addp
Repo2\ Peca:\ Sublista:> Digite o codigo da peca: 4
Repo2\ Peca:\ Sublista:> Digite o nome da peca: p4
Repo2\ Peca:\ Sublista:> Digite a descricao da peca: p4p4
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo2\ Peca:\ Sublista:4> terminate
Insercao concluida!

Repo2\ Peca:\ Sublista:> listp
Codigo: 4 Nome: p4 Descricao: p4p4
Total de pecas: 1

Repo2\ Peca:\ Sublista:> addp
Repo2\ Peca:\ Sublista:> Digite o codigo da peca: 5
Repo2\ Peca:\ Sublista:> Digite o nome da peca: p5
Repo2\ Peca:\ Sublista:> Digite a descricao da peca: p5p5
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo2\ Peca:\ Sublista:5> getp 4
Peca recuperada!

Repo2\ Peca:4\ Sublista:5> addsubpart 150
Peca(s) adicionada(s)!

Repo2\ Peca:4\ Sublista:5> clearlist
Lista limpa!

Repo2\ Peca:4\ Sublista:5> terminate
Insercao concluida!

Repo2\ Peca:4\ Sublista:> listp
Codigo: 4 Nome: p4 Descricao: p4p4
Codigo: 5 Nome: p5 Descricao: p5p5
Total de pecas: 2

Repo2\ Peca:4\ Sublista:> getp 5
Peca recuperada!

Repo2\ Peca:5\ Sublista:> showp
Codigo: 5 Nome: p5 Descricao:p5p5
Peca primitiva

Repo2\ Peca:5\ Sublista:>
```

10 – Adicionando uma subpeça duas vezes:

```
Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 1
Repo1\ Peca:\ Sublista:> Digite o nome da peca: p1
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: p1p1
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:1> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 2
Repo1\ Peca:\ Sublista:> Digite o nome da peca: p2
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: p2p2
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:2> getp 1
Peca recuperada!

Repo1\ Peca:1\ Sublista:2> addsubpart 20
Peca(s) adicionada(s)!

Repo1\ Peca:1\ Sublista:2> addsubpart 50
Peca(s) adicionada(s)!

Repo1\ Peca:1\ Sublista:2> terminate
Insercao concluida!

Repo1\ Peca:1\ Sublista:> getp 2
Peca recuperada!

Repo1\ Peca:2\ Sublista:> showp
Codigo: 2 Nome: p2 Descricao:p2p2
Peca agregada, subcomponentes:
    Codigo: 1 Nome: p1 Descricao: p1p1 [Quantidade = 70]

Repo1\ Peca:2\ Sublista:> _
```

11 – Exemplo de dois clientes (I)

Um sistema distribuído deve permitir que múltiplos clientes consigam se conectar ao servidor, permitindo o acesso e mantendo a coerência do sistema. No exemplo a seguir, temos dois clientes se conectando a um repositório de peças. O cliente A (imagem à esquerda) lista os repositórios, adiciona a peça de código 1 e lista as peças. Após efetuadas as ações do cliente A, o cliente B (imagem à direita) lista as peças e encontra a peça criada pelo cliente A; em seguida adiciona a peça de código 2 com 399 subpartes da peça 1. O cliente A percebe a alteração ao conseguir acessar a peça de código 2.

Repare que mesmo quando um dos clientes fecha a conexão (através do comando ctrl+c no terminal ou o comando “quit” no console do Java), o servidor ainda está ativo, permitindo que os dados das peças “persistam” e possam ser observados por outros clientes.

```
Prompt de Comando - java -classpath C:\Users\andre\Desktop\ Executar.Client
C:\Users\andre\Desktop>java -classpath C:\Users\andre\Desktop\ Executar.Client
Repo1\ Peca:\ Sublista:> listrepos

Repo1
Repo2
Repo3

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 1
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca 1
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: peca de codigo 1(um)
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:1> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Total de pecas: 1

Repo1\ Peca:\ Sublista:> getp 2
Peca recuperada!

Repo1\ Peca:2\ Sublista:> showp
Codigo: 2 Nome: peca 2 Descricao:peca de codigo 2(dois)
Peca agregada, subcomponentes:
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um) [Quantidade = 399]

Repo1\ Peca:2\ Sublista:>

Prompt de Comando - java -classpath C:\Users\andre\Desktop\ Executar.Client
C:\Users\andre>java -classpath C:\Users\andre\Desktop\ Executar.Client
Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Total de pecas: 1

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 2
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca 2
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: peca de codigo 2(dois)
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:2> getp 1
Peca recuperada!

Repo1\ Peca:1\ Sublista:2> addsubpart 399
Peca(s) adicionada(s)!

Repo1\ Peca:1\ Sublista:2> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Total de pecas: 1

Repo1\ Peca:1\ Sublista:2> showp
Codigo: 1 Nome: peca 1 Descricao:peca de codigo 1(um)
Peca primitiva

Repo1\ Peca:1\ Sublista:2> terminate
Insercao concluida!

Repo1\ Peca:1\ Sublista:> getp 2
Peca recuperada!

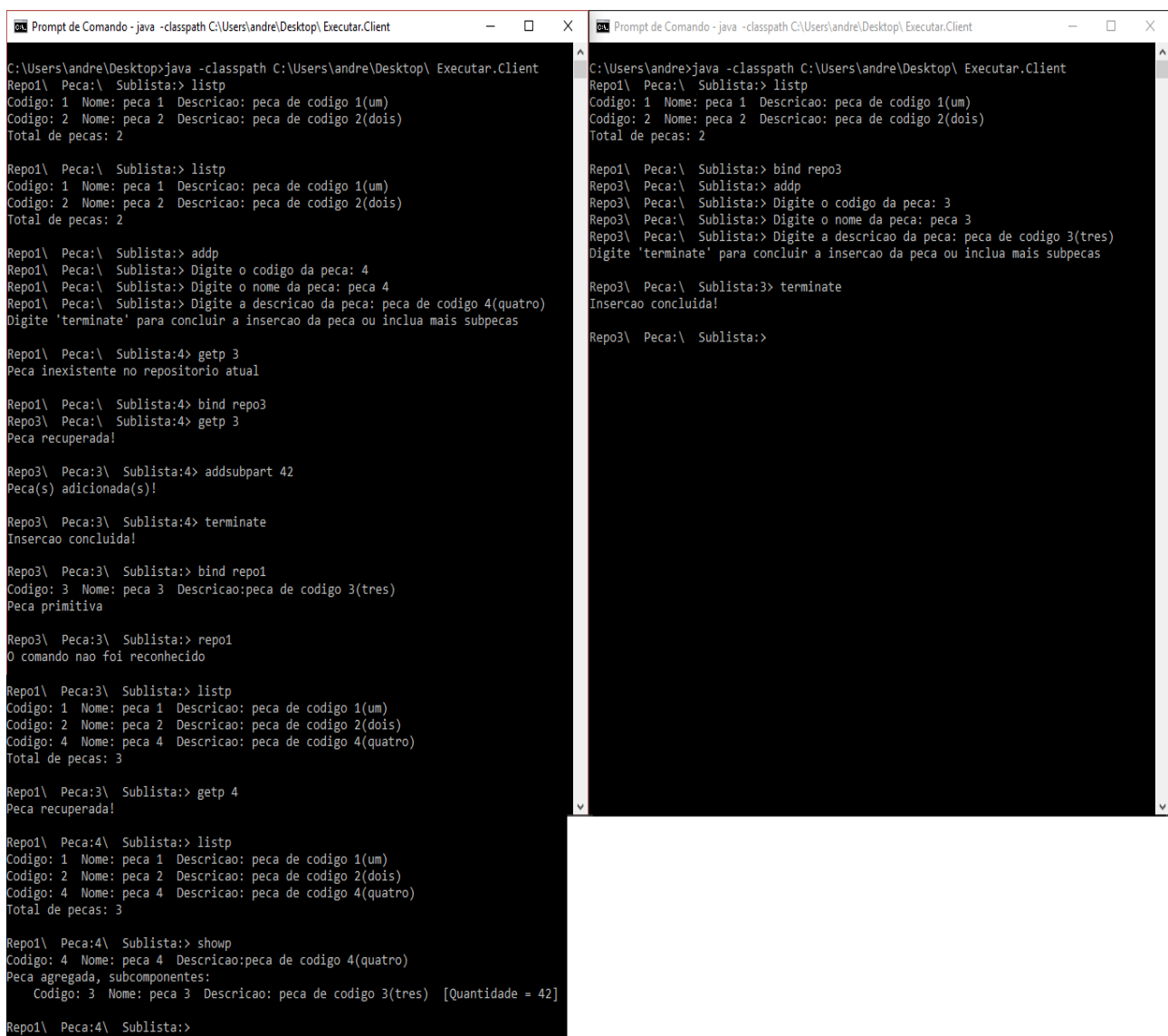
Repo1\ Peca:2\ Sublista:> showp
Codigo: 2 Nome: peca 2 Descricao:peca de codigo 2(dois)
Peca agregada, subcomponentes:
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um) [Quantidade = 399]

Repo1\ Peca:2\ Sublista:>
```


12 – Exemplo de dois clientes (II)

Neste caso, dois clientes se conectam a repositórios diferentes. Ambos podem utilizar peças agregadas e primitivas de outros repositórios. Assim como no exemplo anterior, o cliente A e B listam as peças persistidas no servidor após realizarem o passo anterior (teste de número 11) e se desconectarem do servidor.

O cliente B acessa o repositório 3 e cria uma peça primitiva de código 3. Posteriormente, o cliente A cria uma peça de código 4, acessando o repositório 3 para adicionar subpartes da peça de código 3 à peça de código 4. Confirma-se a integridade do sistema ao recuperarmos, através do cliente A, a peça 4 e executarmos o comando “showp” no final:



```
Prompt de Comando - java -classpath C:\Users\andre\Desktop\ Executar.Client
C:\Users\andre\Desktop>java -classpath C:\Users\andre\Desktop\ Executar.Client
Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Codigo: 2 Nome: peca 2 Descricao: peca de codigo 2(dois)
Total de pecas: 2

Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Codigo: 2 Nome: peca 2 Descricao: peca de codigo 2(dois)
Total de pecas: 2

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 4
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca 4
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: peca de codigo 4(quatro)
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:4> getp 3
Peca inexistente no repositorio atual

Repo1\ Peca:\ Sublista:4> bind repo3
Repo3\ Peca:\ Sublista:4> getp 3
Peca recuperada!

Repo3\ Peca:3\ Sublista:4> addsubpart 42
Peca(s) adicionada(s)!

Repo3\ Peca:3\ Sublista:4> terminate
Insercao concluida!

Repo3\ Peca:3\ Sublista:> bind repo1
Codigo: 3 Nome: peca 3 Descricao:peca de codigo 3(tres)
Peca primitiva

Repo3\ Peca:3\ Sublista:> repo1
O comando nao foi reconhecido

Repo1\ Peca:3\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Codigo: 2 Nome: peca 2 Descricao: peca de codigo 2(dois)
Codigo: 4 Nome: peca 4 Descricao: peca de codigo 4(quatro)
Total de pecas: 3

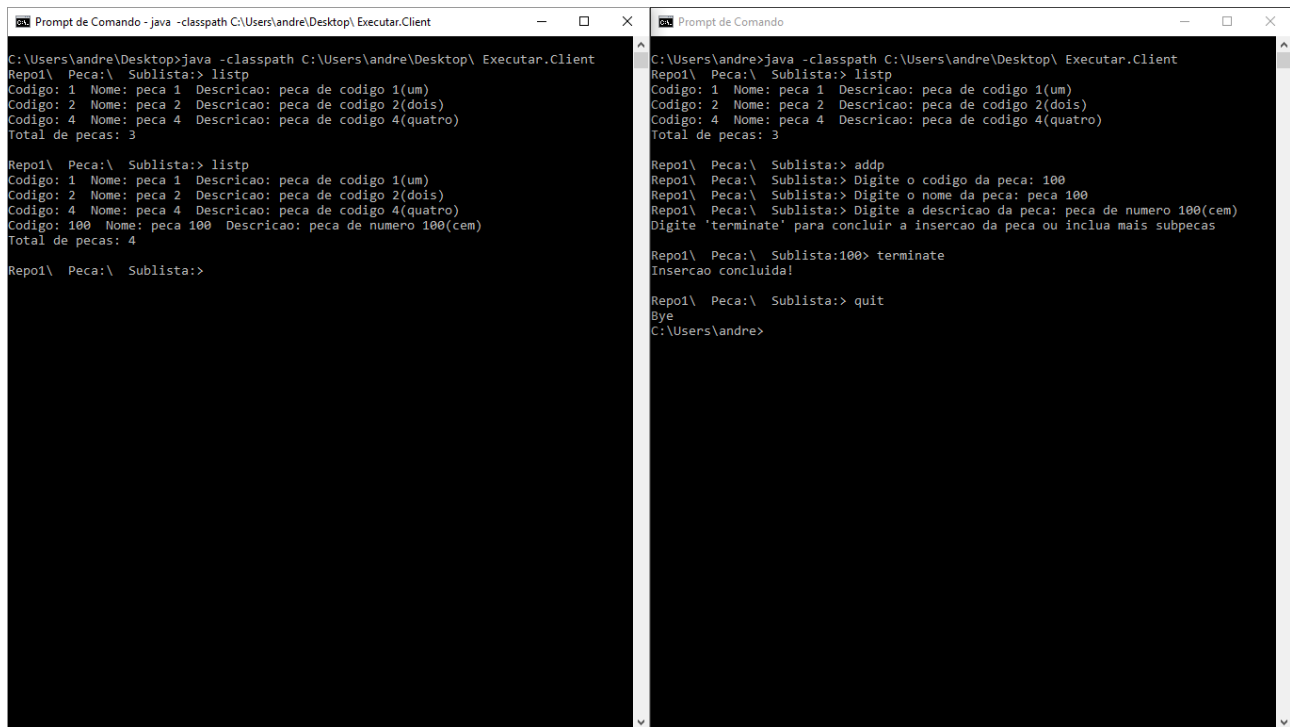
Repo1\ Peca:3\ Sublista:> getp 4
Peca recuperada!

Repo1\ Peca:4\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Codigo: 2 Nome: peca 2 Descricao: peca de codigo 2(dois)
Codigo: 4 Nome: peca 4 Descricao: peca de codigo 4(quatro)
Total de pecas: 3

Repo1\ Peca:4\ Sublista:> showp
Codigo: 4 Nome: peca 4 Descricao:peca de codigo 4(quatro)
Peca agregada, subcomponentes:
Codigo: 3 Nome: peca 3 Descricao: peca de codigo 3(tres) [Quantidade = 42]
Repo1\ Peca:4\ Sublista:>
```

13 – Exemplo de dois clientes (III)

Fica claro que o sistema permanece consistente após as modificações feitas pelo cliente B, ao adicionar uma peça de código 100 e desconectar-se. Desta forma o cliente A consegue observar através do comando “listp” a nova peça inserida.



```
C:\Users\andre\Desktop>java -classpath C:\Users\andre\Desktop\ Executar.Client
Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Codigo: 2 Nome: peca 2 Descricao: peca de codigo 2(dois)
Codigo: 4 Nome: peca 4 Descricao: peca de codigo 4(quatro)
Total de pecas: 3

Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Codigo: 2 Nome: peca 2 Descricao: peca de codigo 2(dois)
Codigo: 4 Nome: peca 4 Descricao: peca de codigo 4(quatro)
Codigo: 100 Nome: peca 100 Descricao: peca de numero 100(ce)
Total de pecas: 4

Repo1\ Peca:\ Sublista:>

C:\Users\andre>java -classpath C:\Users\andre\Desktop\ Executar.Client
Repo1\ Peca:\ Sublista:> listp
Codigo: 1 Nome: peca 1 Descricao: peca de codigo 1(um)
Codigo: 2 Nome: peca 2 Descricao: peca de codigo 2(dois)
Codigo: 4 Nome: peca 4 Descricao: peca de codigo 4(quatro)
Total de pecas: 3

Repo1\ Peca:\ Sublista:> addp
Repo1\ Peca:\ Sublista:> Digite o codigo da peca: 100
Repo1\ Peca:\ Sublista:> Digite o nome da peca: peca 100
Repo1\ Peca:\ Sublista:> Digite a descricao da peca: peca de numero 100(ce)
Digite 'terminate' para concluir a insercao da peca ou inclua mais subpecas

Repo1\ Peca:\ Sublista:100> terminate
Insercao concluida!

Repo1\ Peca:\ Sublista:> quit
Bye
C:\Users\andre>
```

7) Referências Bibliográficas

1-) Getting Started Using Java™ RMI

<http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html> – 2016