UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Pablo Ferreira - 3480
Samuel Sena - 3494

TRABALHO PRÁTICO Parte 4
Java RMI
SISTEMAS DISTRIBUÍDOS - CCF 355

Florestal
2021

Introdução

Nesta parte do trabalho, foi implementado o sistema distribuído de álbum de figurinha Alpokebum utilizando a linguagem de programação Java e seu Middleware RMI embutido. O sistema trabalha com a arquitetura cliente-servidor, sendo que o servidor mantém armazenado objetos persistentes em disco que contém toda as informações dos usuários cadastrados, bem como todos os itens que cada um possui (figurinhas, álbum e quantidade de coins). Além disso, também é armazenado pelo servidor um objeto que contém todas as informações de vendas de figurinhas cadastradas pelos usuários e que se encontram disponíveis para serem compradas por quem se interessar no sistema.

Para executar toda a aplicação, os dois arquivos de código fonte “RMIServer.java” e “RMIClient.java” presentes dentro do projeto do NetBeans devem ser executados, sendo que primeiro deve ser executado o “RMIServer” e em seguida o “RMIClient”. A execução do “RMIServer” necessita ser interrompida caso deseje-se encerrar por completo o programa, visto que o mesmo aceita inúmeras conexões de diferentes instâncias de execução do “RMIClient” em sequência. Escolhemos essa forma de funcionamento para que inúmeras conexões com diferentes usuários possam acontecer de maneira sequencial sem a necessidade de reiniciar ambos os programas. Lembrando que caso seja escolhido a opção de finalizar durante a execução do “RMIClient” o mesmo deverá ser reexecutado caso seja desejado uma nova conexão.

Nesta parte do trabalho, escolhemos por não implementar ainda uma interface gráfica, nos concentrando apenas em uma implementação eficiente e satisfatória do back-end do sistema do álbum de figurinhas. Devido a isso, a forma como as figurinhas são representadas no álbum é: 0 para figurinha não colada e espaço disponível para colá-la, e 1 para figurinha presente (já colada) no álbum, não sendo possível colá-la novamente.

Desenvolvimento

O desenvolvimento desta etapa do trabalho se iniciou partindo do código entregue na etapa passada (API de Sockets), aproveitamos todas as classes

auxiliares usadas como estruturas de dados e formas de abstração do sistema do álbum de figurinhas Alpokebum, que são elas:

- Classe PacoteFigurinha:
    - Classe responsável pela representação de um pacote de figurinha com 4 figurinhas aleatórias e de valor 5 coins.

Figura 1

```
public class PacoteFigurinha implements Serializable {
    private int Pacote[];
    public PacoteFigurinha(){
        this.Pacote = new int[4];
        Random Rand = new Random();
        for (int i=0;i<4;i++){
            this.Pacote[i]=Rand.nextInt(151);
            //System.out.println("Fig: "+ Pacote[i]);
        }
    }
    public int [] GetFigurinhas(){
        return this.Pacote;
    }
}
```

Classe PacoteFigurinha

- Classe AlbumFigurinha:
    - Classe responsável pela representação lógica do álbum de figurinhas do Pokemon, sendo responsável por armazenar um vetor inteiro com 151 posições (1 para figurinha colada e 0 para figurinha não colada) e todas operações em cima destes dados.

Figura 2

```
public class AlbumFigurinhas implements Serializable {
    private int matriz[];
    public AlbumFigurinhas(){
        this.matriz = new int[151];
        for (int i=0;i<151;i++){
            this.matriz[i]=0;
        }
    }
```

Classe AlbumFigurinha

- Classe Usuario:
    - Classe responsável por armazenar todos os atributos e operações que um usuário possui e pode executar dentro do sistema de álbum de figurinha. São armazenados informações de login, álbum de figurinha do usuário, quantidade de coins e um vetor de figurinhas disponíveis para serem coladas que o usuário possui.

Figura 3

```
public class Usuario implements Serializable{
    private String nome_de_usuario;
    private String senha;
    private AlbumFigurinhas album;
    private float coins;
    private ArrayList<Integer> figurinhas_sem_colar;

    public Usuario(String nomeusuario,String senha){
        this.nome_de_usuario = nomeusuario;
        this.senha = senha;
        this.album = new AlbumFigurinhas();
        this.coins = 0;
        this.figurinhas_sem_colar = new ArrayList();
    }
```

Classe Usuario

- Classe Persistencia
    - Classe apenas com métodos estáticos responsável por realizar a leitura e gravação de objetos que se deseja armazenar de maneira persistente em disco. Vale ressaltar que para a utilização dos métodos dessa classe com algum objeto, o mesmo deverá ser uma implementação da interface *Serializable.*

- Classe VendaFigurinha:
    - Classe responsável por armazenar todas as figurinhas disponíveis para negociação. Para isso, a mesma armazena informações sobre quais figurinhas estão disponíveis, além de o respectivo preço e dono atual de cada uma (para uma futura compensação na carteira do vendedor caso a figurinha seja vendida).

Figura 4

```
public class VendaFigurinha implements Serializable{
    private static final long serialVersionUID = 1L;
    private ArrayList<Integer> Lista_Figurinhas;
    private ArrayList<Float> Valor_Figurinhas;
    private ArrayList<String> Proprietarios;
    boolean NecessitaCompensacao;
    private String Nome_Vendedor_a_Compensar;
    private float Valor_a_Compensar;

    public VendaFigurinha(){
        this.Lista_Figurinhas = new ArrayList();
        this.Valor_Figurinhas = new ArrayList();
        this.Proprietarios = new ArrayList();
        this.NecessitaCompensacao = false;
        this.Nome_Vendedor_a_Compensar = null;
        this.Valor_a_Compensar = 0;
    }
```

Classe VendaFigurinha

Implementação do Java RMI

Uma vez que tínhamos conhecimento do pleno funcionamento das classes utilizadas para armazenar, realizar operações, salvar de forma persistente e simbolizar o álbum de figurinhas, as figurinhas e as vendas de figurinhas, partimos para os preparativos para utilizar o middleware Java RMI. A primeira etapa foi a declaração da nossa interface remota chamada “InterfaceFigurinhas”, que estende a interface “Remote” (para assim ser reconhecida pela linguagem como uma interface remota). Esta apresenta todos os métodos disponíveis para invocação remota por parte do cliente.

Figura 5

```
public interface InterfaceFigurinhas extends Remote {
    public Usuario loga_usuario(String nome,String senha) throws RemoteException;
    public boolean cria_usuario(String nome,String senha) throws RemoteException;
    public boolean atualiza_usuario(Usuario a) throws RemoteException;
    public VendaFigurinha get_vendas() throws RemoteException;
    public boolean atualiza_vendas(VendaFigurinha a) throws RemoteException;
}
```

InterfaceFigurinhas

É possível identificar na figura 5 todos os métodos necessários para a comunicação satisfatória entre cliente e servidor no nosso sistema do Alpokebum.

Em seguida realizamos a declaração da classe “RMIServerImple” que estende a classe “UnicastRemoteObject” e implementa a interface “InterfaceFigurinhas” e tem como principal objetivo realizar a implementação de todos os métodos definidos nela, além de permitir a inicialização do middleware RMI.

Figura 6

```
public class RMIServerImple extends UnicastRemoteObject implements InterfaceFigurinhas{
    ArrayList <Object> ListaUsuarios = null;
    ArrayList <Object> LVendas = null;

    protected RMIServerImple() throws RemoteException{
        super();
        ListaUsuarios = Persistencia.lerArquivoBinario("Persistencia.txt");
        LVendas = Persistencia.lerArquivoBinario("VendasFigurinha.txt");
        if(LVendas.size()<1){
            LVendas.add(new VendaFigurinha());
        }
        ((VendaFigurinha)LVendas.get(0)).PrintaFigurinhasAVenda();
    }
```

Classe RMIServerImple

Lembrando que todos os métodos presentes na interface “InterfaceFigurinhas” foram implementados na classe acima, obedecendo procedimentos para garantir a integridade dos dados dos usuários e vendas.

Em sequência, implementamos o executável “RMIServer.java”. Nele realizamos uma instanciação de um objeto “RMIServerImple” atrelado ao servidor RMI com nome de “server” e também iniciamos um registro na porta 1099. Dessa forma, o nosso servidor que utiliza o middleware RMI estava pronto e aguardando conexões de clientes.

Figura 7

```
public class RMIServer {

    public static void main(String args[])
    {
        try {
            Registry reg = LocateRegistry.createRegistry(1099);
            System.out.println("Server do Alpokebum ativo!");
            Naming.rebind("server", new RMIServerImple());
        }
        catch (Exception e) {
            System.out.println("error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Executável RMIServer

E por último, realizamos a implementação do executável “RMIClient.java”. Inicialmente localizamos o registro do servidor RMI no ip 127.0.0.1 (localhost) e porta 1099 e em seguida instanciamos o objeto remoto “RMI” do tipo InterfaceFigurinhas localizado no registro com nome de “server”. Dessa maneira já possuímos cliente e servidor conectáveis e possíveis de invocar métodos de maneira completamente remota através do middleware Java RMI.

Figura 8

```
public class RMIClient {
     public static void main(String[] args)
    {
        try
        {
            Registry reg = LocateRegistry.getRegistry("127.0.0.1",1099);
            InterfaceFigurinhas RMI = (InterfaceFigurinhas) reg.lookup("server");
            Scanner sc = new Scanner(System.in);
            Usuario user = null;
            VendaFigurinha Vendinha = null;
            System.out.println("\tOla, bem-vindo ao cliente do Alpokebum!");
```

Executável RMIClient

Em sequência realizamos uma implementação equivalente a etapa anterior do trabalho para o menu interativo do cliente, porém dessa vez, fazendo o uso de métodos remotos localizados no servidor.

Figura 9

```
String nome;
String senha;
System.out.print("Entre com o nome de usuario: ");
nome = sc.nextLine();
System.out.print("Entre com a senha:");
senha = sc.nextLine();
user = RMI.loga_usuario(nome, senha);

if(user != null){
    System.out.println("Usuario logado com sucesso!");
}
else{
    System.out.println("Falha! Nome ou senha errado!");
}
```

Exemplo de chamada de método remoto “loga_usuario()” em RMIClient.

Figura 10

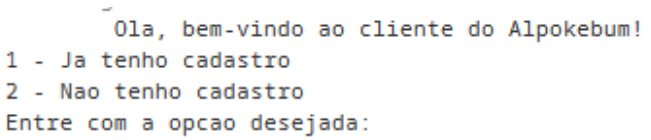
```
while(true){
    if(user==null){break;}
    System.out.println("\tMenu principal");
    System.out.println("1 - Imprimir meu album");
    System.out.println("2 - Comprar Coins");
    System.out.println("3 - Comprar pacotes de figurinhas");
    System.out.println("4 - Colar figurinhas");
    System.out.println("5 - Colar todas figurinhas disponiveis");
    System.out.println("6 - Vender figurinhas no mercado");
    System.out.println("7 - Comprar figurinhas no mercado");
    System.out.println("8 - Imprimir inventario"); //figurinhas sem
    System.out.println("9 - Finalizar");
    System.out.print("Entre com a opcao desejada: ");
    int opcao = 0;
    try{
        opcao= sc.nextInt();
         sc.nextLine();
    }catch(Exception e){
        System.out.println("Erro! Entrada invalida!");
    }
    if(opcao==9){
        RMI.atualiza_usuario(user);
        System.out.println("Finalizando cliente do Alpokebum...");
        //user.PrintaFigurinhas();
        break;
    }
```

Menu interativo em RMIClient e chamada de método remoto “atualiza_usuario()”

Execução

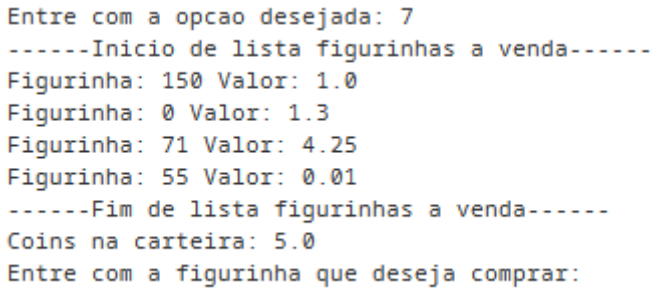
Nesta seção trouxemos algumas figuras que exibem de maneira rápida o funcionamento do algoritmo. Lembramos que os arquivos entregues estarão com a memória persistente “zerada”, sendo necessário criar novos usuários e realizar modificações que serão armazenadas de maneira persistente. E para executar, basta abrir o projeto “RMIFigurinhas” pelo NetBeans e executar os arquivos “RMIServer.java” e “RMIClient.java” do pacote “rmifigurinhas”.

Figura 11



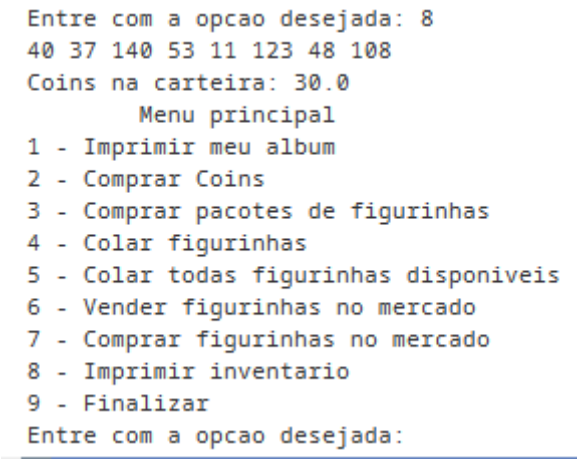
Menu inicial do cliente

Figura 12



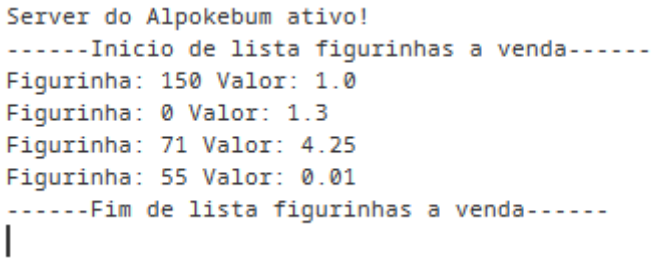
Processo de compra de figurinha.

Figura 13



Operação de impressão de inventário e menu principal.

Figura 14



Tela inicial do servidor

Conclusão

A realização deste trabalho foi de suma importância para o aprendizado e compreensão do funcionamento do middleware Java RMI. Acreditamos que o resultado final cumpre bem a proposta inicial feita por nós, apesar de não possuir uma interface gráfica ainda.

A grande maioria das dificuldades enfrentadas foram facilmente superadas através de pesquisas em fóruns e na documentação da linguagem, além da grande dica dada pelo nosso colega João Arthur no grupo da disciplina no Whatsapp.

Referências

SERIALIZAÇÃO de Objetos em Java. Disponível em: https://www.devmedia.com.br/serializacao-de-objetos-em-java/23413. Acesso em: 28 set. 2021.

INTRODUÇÃO a serialização de objetos. Disponível em: https://www.devmedia.com.br/introducao-a-serializacao-de-objetos/3050. Acesso em: 28 set. 2021.

TUTORIAL RMI - Remote Method Invocation. Disponível em: https://www.devmedia.com.br/tutorial-rmi-remote-method-invocation/6442. Acesso em: 01 out. 2021.