Teoria: Processamento de Imagens

Daniel F. Costa

Departamento de Ciência da Computação
Universidade do Estado do Rio Grande do Norte (UERN) – Natal, RN – Brasil

daniel.ferreira.costa@hotmail.com

1. Código dos ingressos para o filme

```
1  public class Bilheteria extends Thread{
2
3      public static int NIngressos=0;
4
5      public static void main(String[] args) throws Exception{
6
7          System.out.println("Numeros de Cartelas Inicial: 0");
8
9          Bilheteria Fabrica = new Bilheteria("Fabrica");
10         Bilheteria Comprador1 = new Bilheteria("Comprador1");
11         Bilheteria Comprador2 = new Bilheteria("Comprador2");
12         Bilheteria Comprador3 = new Bilheteria("Comprador3");
13         Bilheteria Comprador4 = new Bilheteria("Comprador4");
14         Bilheteria Comprador5 = new Bilheteria("Comprador5");
15
16         Fabrica.start();
17         Comprador1.start();
18         Comprador2.start();
19         Comprador3.start();
20         Comprador4.start();
21         Comprador5.start();
22
23     }
24     public Bilheteria(String id) throws InterruptedException{
25
26         super(id);
27
28     }
29     public void run(){
30
31         while(NIngressos<15){
32
33             if(getName()=="Fabrica")
34                 Bilheteria.metodo(5, getName());
35             else
36                 Bilheteria.metodo(-1, getName());
37
38             try {Thread.sleep(1000);}
39             catch (InterruptedException e){e.printStackTrace();}
40
41         }
42
43     }
44     public synchronized static void metodo(int num, String id){
45
46         if(NIngressos>0 || num==5)
47             NIngressos+=num;
48         else
49             System.out.print("SEM INGRESSOS ");
50         System.out.println(id + " " + NIngressos);
51
52     }
53
54 }
```

Inicialmente, vale ressaltar que a problemática da questão diz que o número máximo de ingressos vendidos deve ser 15.

Entretanto eu preferi adotar que podem ser vendidos infinitos ingressos, desde que não acumule 15 ingressos sem serem vendidos na bilheteria. Conforme a linha 31. TUDO FOI FEITO EM UMA ÚNICA CLASSE.

A função **Main** basicamente cria e inicia as threads.

Depois as threads vão para a função **Bilheteria**, que em essência carrega o "id" da thread para a função **run.** É importante dizer que sem a função **run** elas não executariam em paralelo.

Já na função **run**, eu tive que chamar uma outra função (**metodo**) para poder funcionar como um mutex e sincronizar as trheads.

De modo geral a função **run**, passa o parâmetro 5 se o id for igual a "Fabrica" e o parâmetro -1 se for id for "Comprador".

Esse parâmetro é o que vai decidir se vai haver um incremento ou decremento no número de bilhetes na bilheteria na função **metodo**.

2. Código dos ingressos para o filme

```java
public class Main{

    public static void main(String[] args) throws Exception{

        tThreads time1 = new tThreads("Brasil", 1);
        tThreads time2 = new tThreads("Argentina", 0);

        time1.start();
        time2.start();

    }

}
```

Já esse código eu fiz utilizando duas classes, a **Main** e a **tThreads**.

A classe **Main** basicamente cria e inicia as threads.

```java
public class tThreads extends Thread{

    private int l;
    public static int comp=0;

    public tThreads(String nome, int num) throws InterruptedException{

        super(nome);
        l=num;

    }
    public void run(){

        for (int x=l+5; l<x; l++){

            while(getName()=="Brasil" && comp!=0){}
            while(getName()=="Argentina" && comp!=1){}

            System.out.println(l + getName());
            comp++;
            if(comp==2){comp=0;}

            if(getName()=="Argentina" && l==4)
                System.out.println("DONE!!");

            try {Thread.sleep(1000);}
            catch (InterruptedException e){e.printStackTrace();}

        }

    }

}
```

É na classe **tThreads** que as coisas realmente acontecem. Essa classe possui duas funções: **tThreads** e **run**.

Em essência **tThreads** apenas carrega a string (Brasil ou Argentina) para a função **run.** Também nessa mesma função a variável global "l" se iguala ao "num" de cada thread, por "l" não ser uma variável do tipo static seu valor não é atualizado e assim pode-se transportar também o parâmetro inteiro (é como se cada thread tivesse acesso ao seu próprio "l").

É importante dizer que sem a função **run** as threads não executariam em paralelo.

Na função **run** a thread Brasil sempre vai executar primeiro, porque a variável global "comp" vai iniciar em zero. Caso a thread Argentina fosse executar primeiro, ela ficaria presa em um loop. Depois de executar, a thread Brasil muda "comp" para 1, dessa vez é Brasil que ficará preso no loop enquanto espera Argentina terminar. Diferente de "l", "comp" é uma variável static, então as duas threads vão ter acesso ao mesmo valor atualizado de "comp".

3. Socket TCP

```java
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.SimpleDateFormat;

public class ServidorTCP{

    public static void main(String[] args) throws IOException{

        ServerSocket welcomeSocket = new ServerSocket(1213);
        System.out.println("Esperando conexao...");

        while(true){

            Socket socketConexao = welcomeSocket.accept();
            System.out.println("Bem-vindo, servidor executando...");

            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
            Date hora = Calendar.getInstance().getTime();
            String horario = sdf.format(hora);

            DataOutputStream envioCliente = new DataOutputStream(socketConexao.getOutputStream());
            envioCliente.writeBytes(horario);

            socketConexao.close();

        }

    }
}
```

O computador abre uma porta e fica ouvindo até alguém tentar se conectar.

Se o objeto for criado a porta foi aberta. Se outro programa tem o controle da porta, o nosso não funciona.

Após abrir, temos que esperar pelo cliente através do método “accept” na linha 15.

Por fim, basta ler todas as informações que o cliente enviar.

```java
import java.io.*;
import java.net.*;

public class ClienteTCP{

    public static void main(String[] args) throws IOException{

        Socket socketCliente = new Socket("localhost", 1213);
        BufferedReader doServidor = new BufferedReader(new InputStreamReader(socketCliente.getInputStream()));

        String hora = doServidor.readLine();

        System.out.println("Hora atual:" + hora);

        socketCliente.close();

    }
}
```

O cliente é ainda mais simples do que o servidor.

Tentar se conectar a porta 1213.

Ler a hora do computador.

4. Socket UDP

```java
import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;
import java.text.SimpleDateFormat;

public class ServidorUDP{

    public static void main(String[] args) throws IOException{

        DatagramSocket socketServidor = new DatagramSocket(1213);
        byte[] recebeDados = new byte[1024];
        byte[] enviaDados = new byte[1024];

        while(true){

            System.out.println("Servidor executando...");
            DatagramPacket recebePacote = new DatagramPacket(recebeDados,recebeDados.length);

            socketServidor.receive(recebePacote);
            InetAddress enderecoIP = recebePacote.getAddress();
            int porta = recebePacote.getPort();

            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
            Date hora = Calendar.getInstance().getTime();
            String horario = sdf.format(hora);

            enviaDados = horario.getBytes();

            DatagramPacket enviaPacote = new DatagramPacket(enviaDados, enviaDados.length, enderecoIP, porta);
            socketServidor.send(enviaPacote);

        }

    }
}
```

O UDP é bem mais confuso que o TCP.

O computador abre uma porta e fica ouvindo até alguém tentar se conectar.

Se o objeto for criado a porta foi aberta. Se outro programa tem o controle da porta, o nosso não funciona.

Após abrir, temos que esperar pelo cliente. Por fim, basta ler todas as informações que o cliente enviar.

```java
import java.io.*;
import java.net.*;

public class ClienteUDP{

    public static void main(String[] args) throws Exception{

        DatagramSocket clienteSocket = new DatagramSocket();
        InetAddress enderecoIP = InetAddress.getByName("localhost");

        byte[] enviaDados = new byte[1024];
        byte[] recebeDados = new byte[1024];

        DatagramPacket recebePacote = new DatagramPacket(recebeDados, recebeDados.length);

        DatagramPacket enviaPacote = new DatagramPacket(enviaDados, enviaDados.length, enderecoIP, 1213);
        clienteSocket.send(enviaPacote);

        clienteSocket.receive(recebePacote);
        String horarioServidor = new String(recebePacote.getData());
        System.out.println("Hora a:" + horarioServidor);
        clienteSocket.close();

    }

}
```

Tentar se conectar a porta 1213. Ler a hora do computador.