

## Parte 2: UDP

### Sumário:

comunicação entre 2 servidores utilizando o serviço UDP . Foi utilizada a linguagem Kotlin e a biblioteca Socket, assim como anteriormente. A porta que o servidor escuta virou “7777/7776/7775” enquanto o IP do cliente é o mesmo da máquina. “127.0.0.1”

Observações: Trabalho não concluído completamente até o prazo. Eu consegui criar um software mais legível, me espelhando em tudo que eu senti falta de ter re-feito na Pt 1, porém algumas decisões custaram caro. O projeto ficou com alguns ‘bugs’ não corrigidos.(A mensagem não é re-enviada caso não recebida e a pesquisa consegue pegar elementos do arquivo em disko mas sempre, por algum motivo, retorna vazio)

### Representação:

Foram implementadas no total 6 Classes: Client, Server, UDPMessenger, Host, Data, Search.

- Client novamente fica responsável por receber input do terminal e transformar em um Objeto(Data ou Message). Além disso ele sabe enviar e receber mensagens.
- Server possui a responsabilidade de enviar/receber mensagens, pesquisar combustível mais barato em um dado raio e salvar informações em um disco txt.
- UDPMessenger é a classe pai de Client e Objeto. Ele possui métodos e atributos compartilhados entre ambas as classes e sua proposta, como esperado, é ser uma classe de abstração da comunicação UDP. Internamente possui os métodos de enviar/receber mensagens e as classes abstratas Data e Message usadas das mensagens
- Host lida com a comunicação entre um servidor e vários clientes. Ele cuida da comunicação entre ambos, quando um precisa enviar o outro precisa estar escutando, e vice-versa. A proposta do Host é dizer quem vai falar e quem irá escutar em cada intervalo
- Data/Search são abstrações das mensagens. Possuem internamente o método toString() que permite que as mensagens sejam “printadas” ou salvas em disco

Assim como a Pt 1 algumas abstrações a mais foram criadas:

MessageTypes[Enum]<sup>1</sup> e Coordinate[DataClass]<sup>2</sup>

1- abstrai tipos de mensagens que são aceitas

2-abstrai uma coordenada(x,y) no plano cartesiano.

principais métodos:

sendUdpMessage -> enviar mensagem

receiveUdpMessage -> receber mensagem

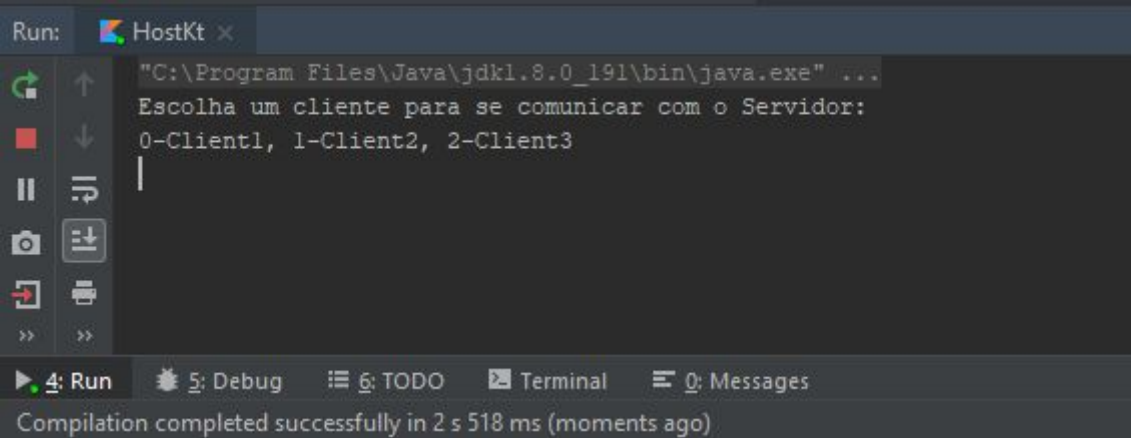
saveData -> salvar informação em disco txt

getData -> ler disco txt e retornar objeto

setupMessage -> criar mensagem

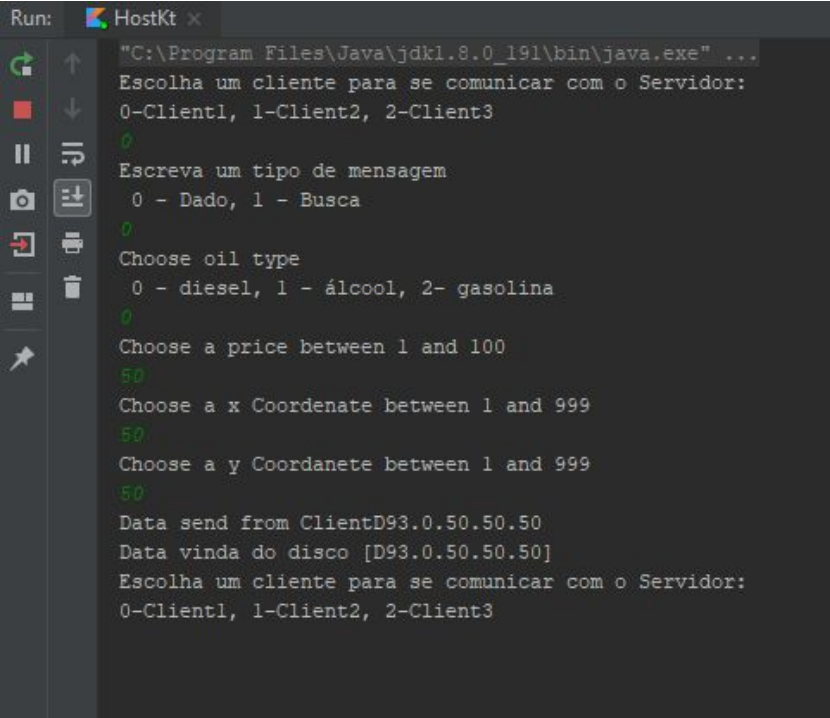
Teste:

Início do programa(escolha um cliente)



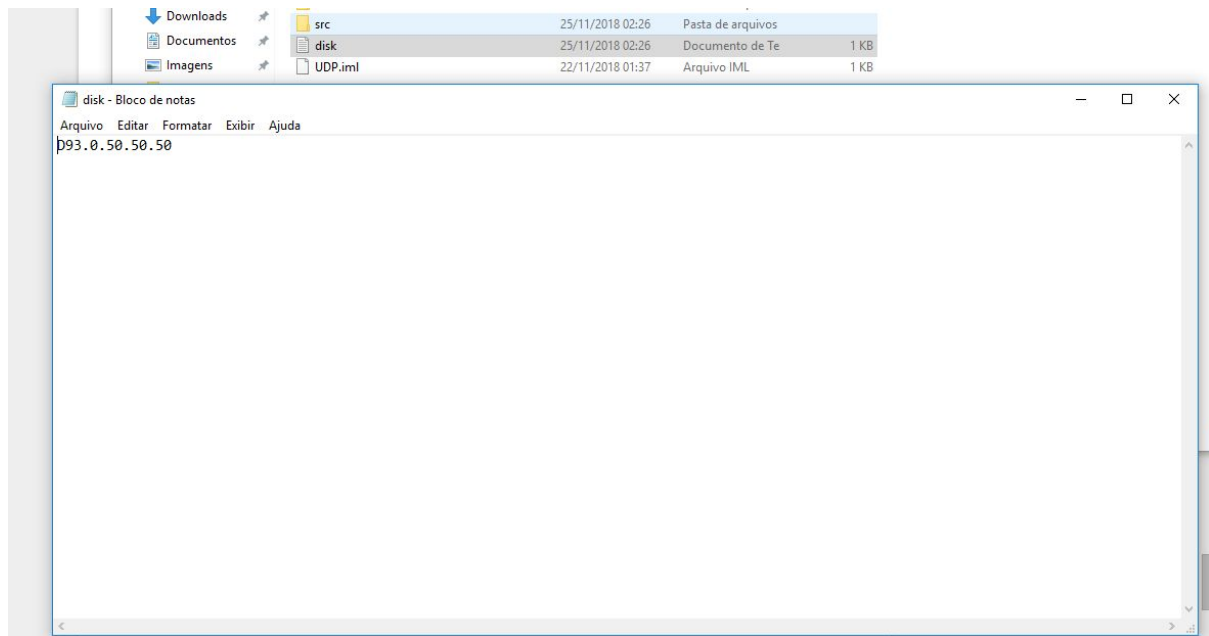
```
Run: HostKt x
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Escolha um cliente para se comunicar com o Servidor:
0-Client1, 1-Client2, 2-Client3
|
4: Run 5: Debug 6: TODO Terminal 0: Messages
Compilation completed successfully in 2 s 518 ms (moments ago)
```

Salvando um dado no Disco:

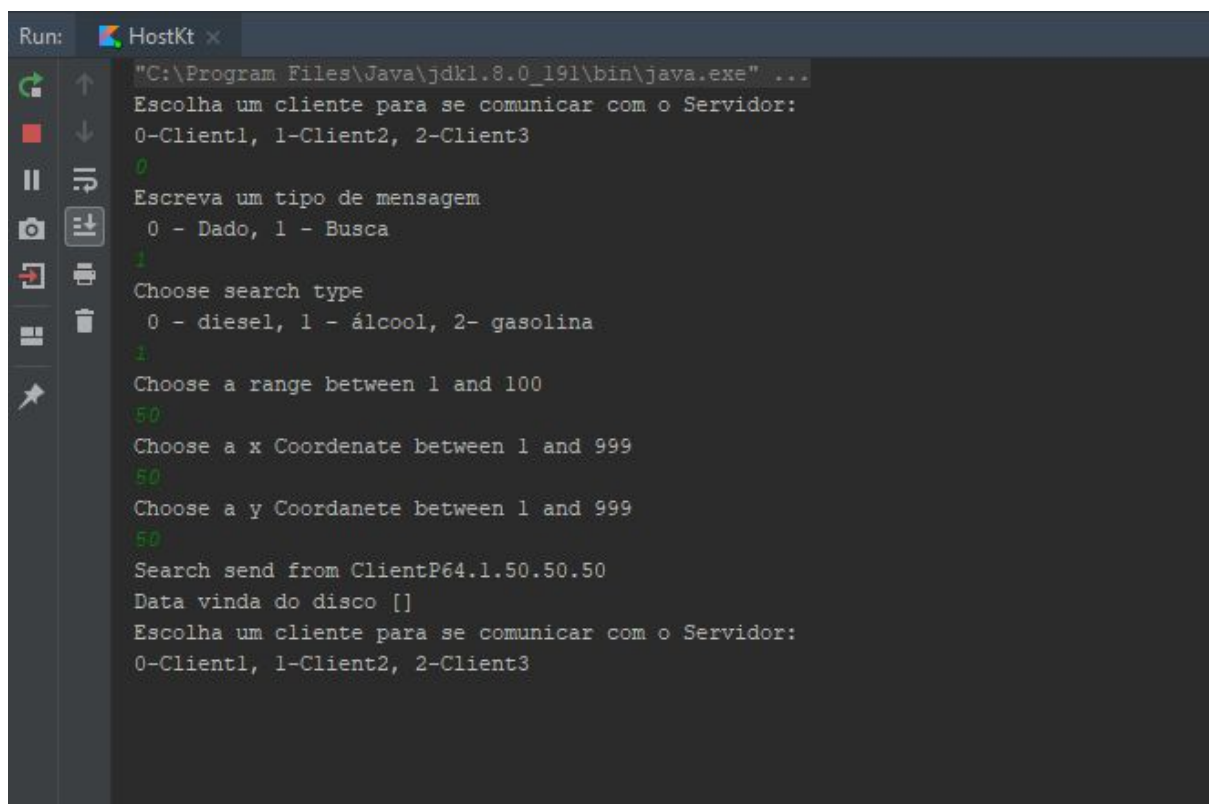


```
Run: HostKt x
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Escolha um cliente para se comunicar com o Servidor:
0-Client1, 1-Client2, 2-Client3
0
Escreva um tipo de mensagem
0 - Dado, 1 - Busca
0
Choose oil type
0 - diesel, 1 - álcool, 2- gasolina
0
Choose a price between 1 and 100
50
Choose a x Coordenate between 1 and 999
50
Choose a y Coordanete between 1 and 999
50
Data send from ClientD93.0.50.50.50
Data vinda do disco [D93.0.50.50.50]
Escolha um cliente para se comunicar com o Servidor:
0-Client1, 1-Client2, 2-Client3
```

Conferindo dado:



Fazendo uma busca:



Conclusão: Como dito previamente, algumas funcionalidades ficaram pendentes. Porém, a comunicação funcionou conforme esperado. Cliente e servidor conseguem trocar mensagens livremente, o programa consegue salvar e extrair informação do disco txt. Mais clientes podem ser roteados, porém a pesquisa ficou realmente em falta