**Enunciado**

Partindo de um Jogo do Galo (Tic Tac Toe) feito para um cenário local, pretende-se desenvolver uma variante do jogo onde a parte computacionalmente mais pesada é realizada por um servidor remoto.

*Sugestão: nas alíneas seguintes, corra os programas cliente e servidor numa máquina não partilhada com outros grupos para evitar conflitos (use uma máquina do laboratório e não o sigma).*

1. Descarregue e descomprima o código fonte da aplicação Jogo do Galo/Tic Tac Toe   
   ([servidor[ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ttt-rmi-server.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ttt-rmi-server.zip), [cliente[ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ttt-rmi-client.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ttt-rmi-client.zip)).
   1. Importe os projectos no eclipse, [seguindo os passos aqui indicados](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/01-tools/eclipse/configure-maven-project/index.html).
   2. No ponto de partida o código está todo centralizado no servidor, estando o cliente vazio.
   3. Estude os principais ficheiros com a implementação do jogo (Game.java e TTT.java).
   4. Compile e experimente o jogo na sua versão centralizada (executando mvn compile exec:java).

1. Pretende-se que a classe TTT.java, que implementa o jogo, passe a ser invocável remotamente.   
   Dessa forma, permitir-se-á que haja um cliente remoto (que possivelmente corre em máquina diferente que a máquina que serve o jogo) que interage com os jogadores e que invoca as funções do servidor via Java RMI (Remote Method Invocation).
   1. Comece por desenhar a interface remota do servidor numa interface chamada TTTService. A interface deve expor todas as funções remotas que o cliente precisa de invocar. Para ser uma interface remota, precisa também de herdar de java.rmi.Remote e cada um dos seus métodos deve lançar uma java.rmi.RemoteException.  
      Consulte o [exemplo de interface remota apresentado no livro](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ShapeList.java) para ajuda.
   2. Transforme a classe TTT para que passe a implementar a interface remota TTTService. Para que instâncias desta classe possam ser objectos remotos, modifique a definição da classe TTT para que ela passe a herdar de java.rmi.server.UnicastRemoteObject e acrescente um construtor que lance excepção RemoteException.  
      Consulte o [exemplo da classe shapeListServant apresentado no livro](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ShapeListServant.java) para ajuda.
   3. No projecto servidor, crie uma nova classe com método main, onde correrá o servidor.   
      No método main deverá:
      1. Instanciar um objecto remoto do tipo TTT
      2. Lançar um rmiregistry (serviço de nomes do RMI) e registar o objecto remoto nesse rmiregistry

Consulte o [exemplo da classe ShapeListServer apresentado no livro](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ShapeListServer.java) para ajuda.  
Não se esqueça de actualizar o ficheiro pom.xml.

* 1. Abra agora o ficheiro fonte da classe ttt.Client.
     1. Baseando-se na classe ttt.Game do projecto do servidor, implemente um cliente remoto que, com base nos comandos recebidos pela consola local, invoca métodos do jogo remoto. Assuma que ambos os jogadores de cada jogo usam o mesmo cliente.  
        Consulte o [exemplo do cliente apresentado no livro](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/04-rmi/ShapeListClient.java) para ajuda.
     2. Não se esqueça de na chamada ao método Naming.lookup, definir correctamente o URL que localiza o objecto, na forma: //host:port/name, em que host e port definem a máquina e o porto onde corre o rmiregistry (respectivamente) onde foi registado o objecto remoto e name é o nome que foi atribuído ao objecto pelo servidor quando chamou rebind.
     3. Adicione o tratamento adequado às excepções lançadas quando acontece algo inesperado numa invocação remota.   
        (Algumas das excepções que podem ocorrer são: java.rmi.UnknownHostException, java.rmi.UnmarshalException, java.rmi.MarshalException, java.rmi.RemoteException ejava.rmi.ConnectException)

1. Experimente lançar o servidor e depois um cliente para jogar.

1. Responda às seguintes questões:
   1. Quando se usa SUN RPC é gerado código para converter os dados de e para um formato de rede. O que acontece quando se usa RMI?
   2. Das classes e interfaces Java que usou, quais as que pertencem apenas ao cliente, apenas ao servidor e a ambos?

O resto do enunciado será entregue na aula. O objectivo será estender a solução resultante do enunciado acima com mais procedimentos ou modificar alguns dos seus procedimentos actuais.

**Entrega da solução**

Fénix, Avaliação, Projetos, **mini Exercício 2 - Java RMI**

A solução completa deverá ser submetida no Fénix **antes do fim da sua aula de laboratório**.  
Trabalhos submetidos depois da hora de fim da aula não serão considerados.

**Ter atenção ao seguinte:**

* Só são aceites trabalhos de estudantes que estiveram presentes no laboratório.
* Deverá incluir um ficheiro respostas.txt com as respostas às perguntas do enunciado do exercício.
* Deverá também incluir um ficheiro instrucoes.txt com resumo da funcionalidade implementada e com instruções para colocar o programa a funcionar como esperado.  
  Por exemplo:
  + A funcionalidade pedida foi total/parcialmente implementada **...**
  + O servidor deve executar com o seguinte comando:  
    mvn install exec:java
  + O cliente deve executar com o comando:  
    mvn compile exec:java
* Assegure-se que a solução é enviada em formato ZIP e que não contém código compilado (faça mvn clean antes de comprimir).