FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Informática

Licenciatura em Engenharia Informática
PROVA DE TESTE PRÁTICO – Sistemas Distribuídos – 1º ciclo
1º Semestre, 2011/2012

NOTAS: Leia com atenção cada questão antes de responder. A interpretação do enunciado de cada pergunta é um factor de avaliação do teste. **O teste é SEM consulta**. **A duração do teste é de 1h30**.
O enunciado contém **6** páginas que devem ser entregues com a resposta ao teste.

NOME: _____ **NÚMERO**.: _____

NOTA IMPORTANTE: as respostas dadas não devem ser relativas à implementação efectuada pelo aluno, mas sim ao cenário apresentado nas perguntas.

1) Suponha que se pretende criar um sistema de partilha de informação sobre eventos com as funcionalidades base do sistema desenvolvido no trabalho prático. Suponha que, quando em funcionamento, o sistema é constituído por um conjunto de clientes/*peers*, um para cada utilizador, e um servidor global.

O servidor global executa numa máquina com endereço conhecido. Este servidor mantém informação sobre os eventos de todas as comunidades e sobre todos os recursos a eles associados. Cada recurso tem um identificador único. Para cada recurso. O servidor mantém uma lista de referências para servidores de recursos a partir dos quais o recurso pode ser obtido.

Os clientes/peers disponibilizam recursos a outros clientes. Para tal, executam um servidor de recursos.

No âmbito deste sistema, responda às seguintes questões

NOTAS: 1) as respostas devem ser baseadas nesta solução e não na que implementou no trabalho prático; 2) ... substitui partes de código omitido por simplicidade ou por a mesmo já ter aparecido anteriormente; 3) **Em cada anexo leia atentamente a sua funcionalidade e o modo como os métodos devem ser invocados nos comentários do código**.

1. Complete o anexo A com o código do FanZoneApp. O método "*doit"* deve criar o servidor de recursos local e registá-lo no servidor de registo RMI sob o nome "ResourceServer". O método getResourceStream deve criar um InputStream para o recurso indicado. O servidor global está acesível no endereço //8.8.8.9/GlobalServer
2. Complete o anexo B com o servidor de recursos baseado em web services (SOAP) – idêntico à classe ResourceServer no anexo A e o código do método "*doit"* que criar o servidor de recursos local. O servidor deve ser acessível a partir do endereço *http://localhost:8080/ResourceServer*.
3. Explique brevemente os passos necessário para criar, em java, um cliente para um serviço REST.

4. O caching é uma técnica tradicional em sistemas distribuídos. Considerando a arquitectura base descrita nesta pergunta, mas a funcionalidade completa do trabalho prático, discuta duas situações em que seria particularmente vantajoso usar caching – explicando porquê e como se geria a cache.

SITUAÇÂO 1:

SITUAÇÂO 2:

2) Para cada pergunta, assinale como **V[erdadeira]** ou **F[alsa]** cada uma das afirmações. **As respostas erradas descontam**.

1. Relativamente a uma referência remota do servidor RMI **Server** que implementa a interface remota **IServer**:
    i. Implementa **Server**;
    ii. Incluí a fábrica de sockets do cliente;
    iii. Garante que, caso não existam falhas de comunicação, o servidor não é recolhido pelo sistema de recolha automática do Java;
    iv. Permite invocar um servidor RMI com uma semântica *at-most-once*.
2. Relativamente ao servidor de registo **rmiregistry**:
    i. Pode incluir o registo do mesmo servidor sob diferentes nomes;
    ii. Permite listar o nome de registo dos servidores registados;
    iii. Permite o registo de um servidor a correr numa máquina diferentes daquela em que o servidor de rmiregistry executa;
    iv. Inclui um mecanismo de controlo de acessos para as pesquisas.
3. Considere um servidor que implementa um serviço remoto usando REST.
    i. Para aceder ao serviço, o cliente cria uma conexão HTTP para o servidor em questão;
    ii. O cliente apenas pode invocar os métodos GET e POST do HTTP para aceder a uma funcionalidade do servidor;
    iii. Um pedido executado por execução do método HTTP POST está sempre completamente definido no URL;
    iv. O resultado é sempre codificado em XML;
4. Considere um serviço implementado usando web services.
    i. O WSDL inclui informação da localização do servidor;
    ii. O WSDL inclui informação do formato e codificação das mensagens trocadas;
    iii. O WSDL é usado pelo programa wsimport para criar as classes que permite a um cliente aceder ao servidor;
    iv. Na implementação standard do Java (JAX-WS) que usou nas aulas, o cliente contacta o servidor usando HTTP.

3) Relativamente ao trabalho prático, responda a **DUAS** das seguintes perguntas (nota: caso responda a mais de duas perguntas, apenas as duas primeiras serão corrigidas).

1. Considere uma arquitectura com um servidor como descrito na figura 1. Discuta os problemas colocados pela falha dum peer, como é que essa falha pode ser detectada e que acções se devem desencadear no servidor.

2. Considere uma arquitectura peer-to-peer, em que, para cada comunidade, cada peer conhece um pequeno número de outros peers da comunidade. Explique como efectuaria uma pesquisa neste sistema.

3. Na opção A.1 pretende-se usar o Google App Engine para executar um servidor do sistema. Explique que servidor colocaria a correr nesta plataforma (i.e., qual a informação que o servidor manteria e qual o protocolo de acesso usado) e qual a vantagem da sua utilização para o bom funcionamento do sistema.

4. Na opção A.5, pretende-se implementar um mecanismo de redundância que garanta que um recurso pouco popular se mantém disponível no sistema. Discuta como poderia implementar esta funcionalidade no contexto de uma arquitectura semelhante à proposta na pergunta 1.

ANEXO A

```
/** Excepção que indica que um ficheiro não existe */
public class NotFoundException extends Exception { ... }

/** Interface do servidor de comunidades */
interface IMainServer
{
  /** devolve array de referências remotas para servidores de recursos
  que têm o recurso indicado */
  public IResourceServer[] getResourceSources( String id)
                        throws                  NotFoundException;
  ...
}

/** Interface do servidor de recursos, existente em cada peer */
interface IResourceServer
{
  /** devolve o conteúdo do recurso indicado. Lança excepção
  NotFoundException se recurso não existir */
  public byte[] getFileContents( String id)
            throws ..., NotFoundException;
  ...
}

/** Classe do servidor de recursos */
class ResourceServer                                    {
  ResourceServer( FanZoneApp app)                       {
    super();
  }

  public byte[] getFileContents(String id)
              throws ..., NotFoundException {
    ...
  }
}
```

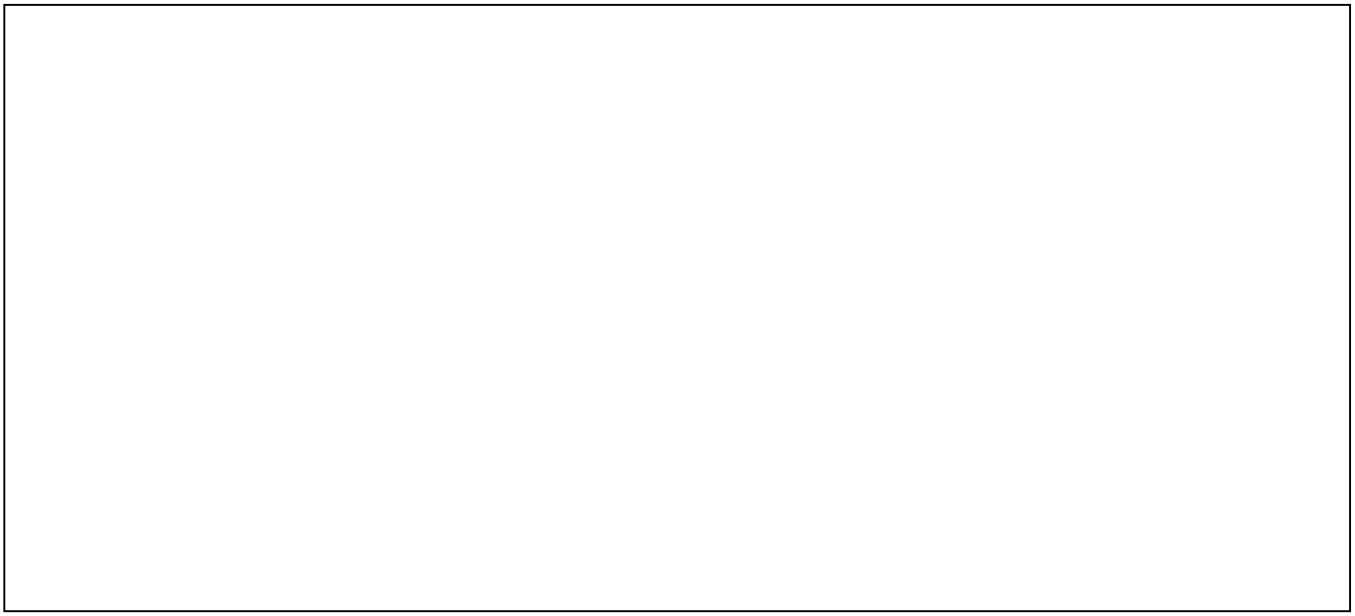
```
/** Classe principal do cliente */
class FanZoneApp implements FanZoneProcessor {
  IResourceServer server;
  ...
  /** Metodo principal: deve iniciar servidor de recursos */
  public void doit() {
    try {
      System.getProperties().put( "java.security.policy", "policy.all");
      if( System.getSecurityManager () == null)
        System.setSecurityManager(                              );
      try {
        LocateRegistry.createRegistry(       );
      } catch( RemoteException e) {
        // do nothing
      }

      server = new ResourceServer( this);
```

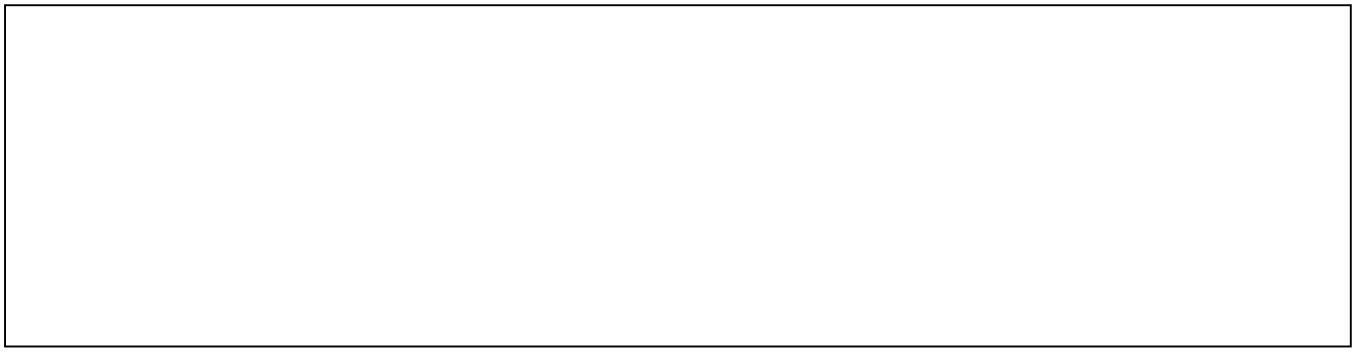
```
    } catch( Exception e) {
       System.out.println( "Impossível iniciar servidor”);
    }
  }
  /** Devolve InputStream para o recurso indicado ou null caso não exista.
   * Use "new ByteArrayInputStream( byte[] arr)" para criar um InputStream
   * a partir dum array de bytes.
   */
  InputStream getResourceStream( String resourceID, ResourceInfo info ) {
```

```
} }
```

ANEXO B



```
/** Classe principal do cliente */
class FanZoneApp implements FanZoneProcessor {
  ...
  /** Metodo principal: deve iniciar servidor de recursos */
  public void doit() {
```



```
  }
}
```