Parte I

1.

a) “rmi” é o nome do protocolo;
“193.127.78.10” é o endereço da máquina em que se encontra a correr o registry rmi;
“5001” é o porto de escuta utilizado pelo registry na máquina;
“cli_serv” é o nome do serviço que se encontra nomeado no registry.

b) Quando é invocado o método lookup pretende-se obter uma referência remota (stub) para o objeto remoto (tanto a referência como objeto implementam a interface remota – define que comportamentos o objeto remoto deverá ter – neste caso só sabemos da existência do int getNClients() mas poderão existir outros métodos) identificado pelo nome cli_serv no registry (serviço de diretório no qual se registam serviços remotos) que se encontra à escuta num socket TCP localizado no porto 5001 da máquina de endereço 193.127.78.10. A mais baixo nível, o que acontece é que o lookup é um método que abre uma conexão TCP com o socket de escuta do registry referido anteriormente (caso falhe, ocorre uma excessão local na máquina que fez o pedido) e faz um pedido com informação suficiente para identificar que se pretende obter um objeto remoto que segue o protocolo rmi de nome “cli_serv”; tanto a informação que é enviada neste pedido como a respetiva resposta (quer seja uma referência para o objeto ou uma excessão do lado do servidor a identificar p.e. que não se encontra nomeado nenhum serviço rmi “cli_serv”) é sempre serializada – significando que não são enviadas verdadeiras referências/endereços mas sim arrays de bytes provenientes da serialização dos objetos.

2.

a)

	Método HTTP	URI	Parâmetros opcionais
1	GET	/students/year/{year}	course
2	GET	/students/{id}	
3	DELETE	/students/{id}	

b)

```
@GetMapping(“/students/year/{year}”)
public List<Student> getStudents(
        @PathVariable(“year”) int year,
        @RequestParam(value=“course”, required=false) String course) {
        /* não é para completar esta parte */
}
```

c) Sabendo que o acesso à API REST em causa deve ser feito através de autênticação básica HTTP e que os pedidos HTTP de um utilizador com username “david” e password “abc” são submetidos incluindo o campo “Authorization: Basic ZGF2aWQ6YWJj” no cabeçalho, a sequência “ZGF2aWQ6YWJj” representa a concatenação do username à password sob o formato “david:abc” codificada com Base64.

Parte II

```
…
// [1]
DatagramPacket dp = new DatagramPacket(new byte[1024], 1024);
ds.receive(dp);
…
```

```
// [2]
try (new ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(dp.getData(), 0,
dp.getLength()))) {
        lInfo = (LocationInfo) ois.readObject();
}
…
// [3]
synchronized (subcribers) {
…
// [4]
int nSubscibers = subcribers.length();
for (int i = 0; I < nSubscribers; i++) {
        SubscriberInfo subsciber = subscribers.get(i);
        if (subscriber.getUserIdToFollow().equals(lInfo.getUserId())) {
                try {
                        subscriber.getOOut().writeUnshared(lInfo);
                } catch (Exception e) {
                        subscriber.getOIn().close();
                        subscriber.getOOut().close();
                        subscribers.remove(subscriber);
                        nSubscribers--;
                        i--;
                }
        }
}
…
// [5]
Socket s = serverSocket.accept();
ObjectInputStream oin = new ObjectInputStream(s.getInputStream());
ObjectOutputStream oout = new ObjectOutputStream(s.getOutputStream());
String userIdToFollow = (String) oin.readObject();
…
// [6]
synchronized (subscribers) {
…
// [7]
udpThread = new UdpThread();
udpThread.setDaemon(true);
udpThread.start();
…
tcpThread = new Thread(new TcpThread());
tcpThread.setDaemon(true);
tcpThread.start();
…
// [8]
Naming.bind(“rmi://193.137.19.19:5002/live_tracking”, service);
…
// [9]
tcpThread.join();
```

```
…
// [10]
public class RemoteService extends UnicastRemoteObject implements RemoteServiceInterface {
…
// [11]
public RemoteService() throws RemoteException {}
…
// [12]
ref.notifyLocationInfo(lInfo);
```