Programação Orientada por Objectos com Java

Ademar Aguiar
www.fe.up.pt/~aaguiar
ademar.aguiar@fe.up.pt

Universidade do Porto
Faculdade de Engenharia
FEUP

Java Networking

java.net.*

Package java.net

- Java dispõe de diversas classes para manipular e processar informação em rede
- São suportados dois mecanismos básicos:
  - Sockets - troca de pacotes de informação
  - URL - mecanismo alto-nível para troca de informação
- Estas classes possibilitam comunicações baseadas em sockets:
  - permitem manipular I/O de rede como I/O de ficheiros
  - os sockets são tratados como *streams* alto-nível o que possibilita a leitura/escrita de/para sockets como se fosse para ficheiros

Package java.net

- O package contém as seguintes classes:
  - URL – encapsula um endereço WWW
  - URLconnection – uma ligação WWW
  - InetAddress – um endereço IP com nome de host
  - Socket – lado do cliente, liga a um porto, utiliza TCP
  - ServerSocket – ausculta um determinado porto por ligações de clientes (a ligação implica TCP)
  - DatagramSocket – um socket UDP, para clientes e servidores
  - DatagramPacket – empacota informação num pacote UDP com informação de encaminhamento IP

Sockets

- Um socket é um mecanismo que permite que programas troquem pacotes de bytes entre si.
- A implementação Java é baseada na da BSD Unix.
- Quando um socket envia um pacote, este é acompanhado por duas componentes de informação:
  - Um endereço de rede que especifica o destinatário do pacote
  - Um número de porto que indica ao destinatário qual o socket usar para enviar informação
- Os sockets normalmente funcionam em pares: um cliente e um servidor

Sockets e Protocolos

- Protocolos *Connection-Oriented*
  - O socket cliente estabelece uma ligação para o socket servidor, assim que é criado
  - Os pacotes são trocados de forma fiável
- Protocolos *Connectionless*
  - Melhor performance, mas menos fiabilidade
  - Exemplos de utilização: envio de um pacote, audio em tempo-real
- Comparação
  - TCP/IP utiliza sete pacotes para enviar apenas um (1/7).
  - UDP utiliza apenas um pacote (1/1).

Sockets/Protocolos em Java

	Cliente	Servidor
Connection-oriented Protocol	Socket	ServerSocket
Connectionless Protocol	DatagramSocket	DatagramSocket

Sockets em Protocolos *Connection-Oriented*

- Pseudo-código típico para um servidor:
  - Criar um objecto ServerSocket para aceitar ligações
  - Quando um ServerSocket aceita uma ligação, cria um objecto Socket que encapsula a ligação
  - O Socket deve criar objectos InputStream e OutputStream para ler e escrever bytes para e da ligação
  - O ServerSocket pode opcionalmente criar um novo *thread* para cada ligação, por forma a que o servidor possa aceitar novas ligações enquanto comunica com os clientes
- Pseudo-código típico para um cliente
  - Criar um objecto Socket que abre a ligação com o servidor, e utiliza-o para comunicar com o servidor

Exemplo: Servidor de Ficheiros

```
public class FileServer extends Thread {
    public static void main(String[] argv) {
        ServerSocket s;
        try {
            s = new ServerSocket(1234, 10);
        }catch (IOException e) {
            System.err.println("Unable to create socket");
            e.printStackTrace();
            return;
        }
        try {
            while (true) {
                new FileServer(s.accept());
            }
        }catch (IOException e) {
        }
    }
    private Socket socket;
    FileServer(Socket s) {
        socket = s;
        start();
    }
}
```


Exemplo: Servidor de Ficheiros...

```
  public void run() {
        InputStream in;
        String fileName = "";
        PrintStream out = null;
        FileInputStream f;
        try {
            in = socket.getInputStream();
            out = new PrintStream(socket.getOutputStream());
            fileName = new DataInputStream(in).readLine();
            f = new FileInputStream(fileName);
        }catch (IOException e) {
            if (out != null)
              out.print("Bad:"+fileName+"\n");
            out.close();
            try {
                socket.close();
            }catch (IOException ie) {
            }
            return;
        }
        out.print("Good:\n");
        // send contents of file to client.
}
```

Exemplo: Servidor de Ficheiros...

```
public class FileClient {
    private static boolean usageOk(String[] argv) {
        if (argv.length != 2) {
            String msg = "usage is: " +
               "FileClient server-name file-name";
            System.out.println(msg);
            return false;
        }
        return true;
    }
    public static void main(String[] argv) {
        int exitCode = 0;
        if (!usageOk(argv))
          return;
        Socket s = null;
        try {
            s = new Socket(argv[0], 1234);
        }catch (IOException e) {
            //...
        }
        InputStream in = null;
        // ...
    }
}
```


Sockets em Protocolos *Connectionless*

- Pseudo-código típico para um servidor:
  - Criar um objecto DatagramSocket associado a um determinado porto
  - Criar um objecto DatagramPacket e pedir ao DatagramSocket para colocar o próximo bloco de dados que recebe no DatagramPacket
- Pseudo-código típico para um cliente
  - Criar um objecto DatagramPacket associado a um bloco de dados, um endereço de destino, e um porto
  - Pedir a um DatagramSocket para enviar o bloco de dados associado ao DatagramPacket para o destino associado ao DatagramSocket
- Exemplo: TimeServer

Objectos URL

- A classe URL fornece um acesso a dados a um mais alto-nível do que os sockets
- Um objecto URL encapsula um *Uniform Resource Locator* (URL) que uma vez criado pode ser usado para aceder a dados de um endereço especificado pelo URL
- O acesso aos dados não necessita de se preocupar com o protocolo utilizado
- Para alguns tipos de dados, um objecto URL sabe devolver os conteúdos. Por exemplo, dados JPEG num objecto ImageProducer, ou texto numa String

Criação de objectos URL

- URL's absolutos

```
try {
   URL js = new URL("http://www.javasoft.com/index.html");
} catch (MalformedURLException e) {
   return;
}
```

- URL's relativos

```
try {
   URL jdk = new URL(js,"java.sun.com/products/JDK/index.html");
}catch (MalformedURLException e) {
    return;
}
```

- Métodos de acesso
  - getProtocol(), getHost(), getFile(), getPort(), getRef(), sameFile(URL), getContent(), openStream()

Fim