Unidad 5. Servicios en red: Clases auxiliares para direccionamiento

Descargar estos apuntes

Índice

Clases auxiliares para direccionamiento

java.net.NetworkInterface

Esta clase representa la interfaz de red, tanto software como hardware, su nombre, la lista de direcciones IP asignadas y toda la información relacionada. Se puede usar en los casos en que queramos usar específicamente una interfaz particular para transmitir nuestro paquete en un sistema con múltiples NIC.

> java.net.InetAddress specification

> **¿Qué es una interfaz de red?**
>
> Se puede pensar en una interfaz de red como un punto en el que su computadora se conecta a la red. No es necesariamente una pieza de hardware, pero también se puede implementar en un software. Por ejemplo, una interfaz de bucle invertido que se utiliza con fines de prueba.

Method	Description
public static Enumeration getNetworkInterfaces()	Devuelve todas las interfaces de red del sistema.
public List getInterfaceAddresses()	Devuelve una lista de todas las direcciones de interfaz en esta interfaz.
public Enumeration getInetAddresses()	Devuelve una enumeración de todas las Inetaddresses vinculadas a esta interfaz de red, si el administrador de seguridad lo permite.
public String getName()	Devuelve el nombre de esta interfaz de red
public int getIndex()	Devuelve el índice asignado a esta interfaz de red por el sistema. Los índices se pueden utilizar en lugar de nombres largos para hacer referencia a cualquier interfaz del dispositivo.
public String getDisplayName()	Este método devuelve el nombre de la interfaz de red en un formato de string legible.
public static NetworkInterface getByName(String name)	Busca y devuelve la interfaz de red con el nombre especificado, o nulo si no existe.
public static NetworkInterface getByIndex(int index)	Realiza una función similar a la función anterior con el índice utilizado como parámetro de búsqueda en lugar del nombre.

Method	Description
public static NetworkInterface getByInetAddress(InetAddress addr)	Este método se usa ampliamente ya que devuelve la interfaz de red a la que está vinculada la dirección de red especificada. Si una InetAddress está vinculada a varias interfaces, se puede devolver cualquiera de las interfaces.
public boolean isUp()	Devuelve un valor booleano que indica si esta interfaz de red está en funcionamiento.

```java
// Java program to illustrate various java.net.NetworkInterface class methods.

public class NetworkInterfaceExample
{
    public static void main(String[] args) throws SocketException,
                                                   UnknownHostException
    {

        // getNetworkInterfaces() returns a list of all interfaces
        // present in the system.
        ArrayList<NetworkInterface> interfaces = Collections.list(
                                            NetworkInterface.getNetworkInterfaces());

        System.out.println("Information about present Network Interfaces...\n");
        for (NetworkInterface iface : interfaces)
        {
            // isUp() method used for checking whether the interface in process
            // is up and running or not.
            if (iface.isUp())
            {
                // getName() method
                System.out.println("Interface Name: " + iface.getName());

                // getDisplayName() method
                System.out.println("Interface display name: " + iface.getDisplayName());

                // getHardwareAddress() method
                System.out.println("Hardware Address: " +
                                Arrays.toString(iface.getHardwareAddress()));

                // getParent() method
                System.out.println("Parent: " + iface.getParent());

                // getIndex() method
                System.out.println("Index: " + iface.getIndex());
                // Interface addresses of the network interface
                System.out.println("\tInterface addresses: ");

                // getInterfaceAddresses() method
                for (InterfaceAddress addr : iface.getInterfaceAddresses())
                {
                    System.out.println("\t\t" + addr.getAddress().toString());
                }
                // Interface addresses of the network interface
                System.out.println("\tInetAddresses associated with this interface: ");

                // getInetAddresses() method returns list of all
                // addresses currently bound to this interface
                Enumeration<InetAddress> en = iface.getInetAddresses();
                while (en.hasMoreElements())
                {
                    System.out.println("\t\t" + en.nextElement().toString());
                }

                // getMTU() method
```

```
                System.out.println("\tMTU: " + iface.getMTU());

                // getSubInterfaces() method
                System.out.println("\tSubinterfaces: " +
                                Collections.list(iface.getSubInterfaces()));

                // isLoopback() method
                System.out.println("\tis loopback: " + iface.isLoopback());

                // isVirtual() method
                System.out.println("\tis virtual: " + iface.isVirtual());

                // isPointToPoint() method
                System.out.println("\tis point to point: " + iface.isPointToPoint());

                // supportsMulticast() method
                System.out.println("Supports Multicast: " + iface.supportsMulticast());

            }
        }

        // getByIndex() method returns network interface
        // with the specified index
        NetworkInterface nif = NetworkInterface.getByIndex(1);

        // toString() method is used to display textual
        // information about this network interface
        System.out.println("Network interface 1: " + nif.toString());

        // getByName() method returns network interface
        // with the specified name
        NetworkInterface nif2 = NetworkInterface.getByName("eth0");
        InetAddress ip = InetAddress.getByName("localhost");

        // getbyInetAddress() method
        NetworkInterface nif3 = NetworkInterface.getByInetAddress(ip);
        System.out.println("\nlocalhost associated with: " + nif3);
    }
}
```

java.net.InterfaceAddress

Esta clase representa una dirección de interfaz de red. Cada dispositivo que tiene una dirección IP tiene una dirección IP en la interfaz de red. De hecho, el comando ping no hace ping a un dispositivo, sino a la dirección de interfaz de los dispositivos.

Java proporciona ciertos métodos para tratar con direcciones de interfaz que se pueden usar en lugares donde existe la necesidad de conocer la topología de la red, para la detección de fallas en una red, etc.

Resumiendo, esta clase representa a una dirección IP, una máscara de red y una dirección broadcast (cuando la dirección es IPv4). Sólo representa una dirección IP address y una longitud de prefijo de red en el caso de direcciones IPv6.

> java.net.InterfaceAddress specification

Method	Description
public InetAddress getAddress()	Devuelve una InetAddress para esta dirección
public InetAddress getBroadcast()	Devuelve InetAddress para la dirección de transmisión para esta dirección de interfaz. Como solo las direcciones IPv4 tienen direcciones de transmisión, se devolvería un valor nulo al usar una dirección IPv6.
public short getNetworkPrefixLength()	Devuelve la longitud del prefijo para esta dirección de interfaz, es decir, la máscara de subred para esta dirección.

```java
// Java program to illustrate methods of java.net.InterfaceAddress class

public class InterfaceaddressExample
{
    public static void main(String[] args) throws SocketException
    {
        // Modify according to your system
        NetworkInterface nif = NetworkInterface.getByIndex(1);
        List<InterfaceAddress> list = nif.getInterfaceAddresses();

        for (InterfaceAddress iaddr : list)
        {
            // getAddress() method
            System.out.println("getAddress() : " + iaddr.getAddress());

            // getBroadcast() method
            System.out.println("getBroadcast() : " + iaddr.getBroadcast());

            // getNetworkPrefixLength() method
            System.out.println("PrefixLength : " + iaddr.getNetworkPrefixLength());

            // hashCode() method
            System.out.println("Hashcode : " + iaddr.hashCode());

            // toString() method
            System.out.println("toString() : " + iaddr.toString());
        }
    }
}
```

java.net.InetAddress

La clase java.net.InetAddress proporciona métodos para obtener la dirección IP de cualquier nombre de host, por ejemplo example www.google.com, www.facebook.com, etc.

> java.net.InetAddress specification

La clase InetAddress se usa para encapsular tanto la dirección IP numérica como el nombre de dominio para esa dirección.

Hay 2 tipos de direcciones:

- Unicast: un identificador para una única interfaz.
- Multicast: un identificador para un conjunto de interfaces.

> ⚠ **Local Name Resolver (hosts file)**
>
> Deberías saber que DNS traduce nombre de dominio en direcciones IP. Pero, ?sabes que hay un archivo en tu sistema que puede sobrescribir esas traducciones?
>
> Es el archivo `hosts` y nos permite mapear nombre de dominio a direcciones IP. Tu archivo HOSTS sólo afecta al comportamiento de tu equipo, por lo que podemos usarlo para crear direcciones personalizadas para IP de nuestra red, o bien para redireccionar / bloquear el acceso a determinados sitios web.
>
> Como puedes imaginar, cambiar de forma incorrecta o **maliciosa** el contenido del archivo hOSTS puede romper fácilmente el comportamiento de tu conexión a Internet, Así que la modificación del archivo no es trivial para los usuarios, algo que es de agradecer.
>
> - Windows
>
> El archivo HOSTS está almacenado como un fichero de texto plano en la carpeta del sistema de Windows.
>
> Abre el menú inicio y escribe "notepad".
>
> Pulsa con el botón derecho y selecciona la opción de "Ejecutar como administrador"
>
> En Notepad, ve a Archivo > Abrir y pega la siguiente ruta:
>
> c:\Windows\System32\Drivers\etc\hosts
>
> Ahora ya puedes editar y guardar los cambios en tu archivo HOSTS.
>
> Para mapear un dominio, añade una nueva línea siguiendo los ejemplos que hay en el archivo.
>
> - OS X & GNU/Linux
>
> El archivo está en /etc/hosts y debes editarlo con privilegios de administrador.

```
# Añadimos las siguientes entradas al archivo hosts
## En clase
##  - como cliente y servidor ponemos la IP de nuestro equipo.
##  - como profesor ponemos la IP del ordenador del profesor
## En casa
##  - como cliente, servidor y profesor ponemos la IP de nuestro equipo.

# En nuestras actividades, para no tener que ir cambiando las direcciones IP, usaremos
# siempre estos nombres de dominio, así los programas funcionarán tanto en clase como
# en casa.
10.100.XX.1 cliente.psp
10.100.XX.1 servidor.psp
10.100.0.1 profesor.psp
```

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	Este método devuelve la instancia de InetAddress que contiene el nombre y la IP del host recibido como parámetro.
public static InetAddress getLocalHost() throws UnknownHostException	Este método devuelve la instancia de InetAddress que contiene el nombre y la IP de LocalHost.
public String getHostName()	Este método devuelve el nombre de host para esta dirección IP.
public String getHostAddress()	Este método obtiene la dirección IP en forma de string.
public boolean isReachable(int timeout)	Este método prueba si esa dirección es accesible.

```java
class InetAddressExample {
    public static void main(String[] args)
        throws UnknownHostException
    {
        // To get and print InetAddress of Local Host
        InetAddress address1 = InetAddress.getLocalHost();
        System.out.println("InetAddress of Local Host : "
                           + address1);

        // To get and print InetAddress of Named Host
        InetAddress address2
            = InetAddress.getByName("45.22.30.39");
        System.out.println("InetAddress of Named Host : "
                           + address2);

        // To get and print ALL InetAddresses of Named Host
        InetAddress address3[]
            = InetAddress.getAllByName("172.19.25.29");
        for (int i = 0; i < address3.length; i++) {
            System.out.println(
                "ALL InetAddresses of Named Host : "
                + address3[i]);
        }

        // To get and print InetAddresses of
        // Host with specified IP Address
        byte IPAddress[] = { 125, 0, 0, 1 };
        InetAddress address4
            = InetAddress.getByAddress(IPAddress);
        System.out.println(
            "InetAddresses of Host with specified IP Address : "
            + address4);

        // To get and print InetAddresses of Host
        // with specified IP Address and hostname
        byte[] IPAddress2
            = { 105, 22, (byte)223, (byte)186 };
        InetAddress address5 = InetAddress.getByAddress(
            "gfg.com", IPAddress2);
        System.out.println(
            "InetAddresses of Host with specified IP Address and hostname : "
            + address5);
    }
}
```

Buscador de equipos (U4S4_HostSeeker)

Tu equipo está conectado a una LAN (Red de Área Local) y probablemente esté usando una dirección IP privada.

Las direcciones pueden ser de clase C (192.168.X.Y), clase B (172.17.X.Y) o clase A (10.X.Y.Z). Eso depende principalmente de la máscara de red y del prefijo de red usado para la configuración del interfaz.

Puedes comprobar esta configuración con los comandos **ifconfig** de OSX GNU/Linux o **ipconfig** de Windows..

Escribe un programa que averigüe, dentro de nuestra red, qué hosts están activos en la red, es decir, qué hosts son "alcanzables" desde tu equipo usando uno de los interfaces disponibles.

Primero vamos a hacerlo de forma sencilla. Escribe un programa que sabiendo la dirección de tu equipo y la longitud del prefijo, pruebe todas las posibles combinaciones.

> Si nuestra IP es 192.168.0.50 y el prefijo es /24, esto indica que los primeros 24 bits de la dirección IP son el identificador de la red, y que los 8 últimos son para identificadores de hosts. Por lo que sólo tenemos que ir probando con los últimos 8 bits (el último dígito de la dirección) para detectar a otros equipos en la red. Esto nos da 254 posibilidades, ya que la 0 y la 255 no se usan para hosts.
>
> Si por el contrario, el prefijo fuese 16, tendríamos que ir cambiando los dos últimos números.
>
> 192.168.0.1 a 192.168.0.254, después 192.168.1.1 a 192.168.1.254, así hasta 192.168.255.1 a 192.168.255.254, es decir, tendría que usar un bucle anidado.

La aplicación sólo debe mostrar la dirección IP de los equipos que sean alcanzables.

Optativo: Una vez que tengas tu aplicación funcionando, intenta hacerla genérica y reutilizable para que funcione en cualquier red, obteniendo el prefijo de red y comprobando todas las posibles direcciones en la red en función del prefijo obtenido.

En ambos casos la aplicación recibirá el nombre de una NIC como argumento y comprobará sólo las direcciones IP asignadas a esa interfaz.

Podemos saber si una dirección es IPv4 o IPv6 usando el operador `instanceof` con las subclases Inet4Address y Inet6Address.

Código del ejemplo

```java
public class U4S4_HostSeeker {

    private final int timeout = 10;
    private String interfaceName;
    private String ipAddress;
    private String networkMask;

    private ArrayList<String> reachableIps = new ArrayList<>();

    public static void main(String[] args)
            throws UnknownHostException, IOException {

        U4S4_HostSeeker seeker = new U4S4_HostSeeker(args);
        System.out.println("Scanning...");
        seeker.processNetwork();
        seeker.listResults();
    }

    public U4S4_HostSeeker(String[] args) throws SocketException, UnknownHostException {
        NetworkInterface networkCard = null;

        switch (args.length) {
            case 1 -> {
                // Get the interface from the name
                interfaceName = args[0];

                networkCard = NetworkInterface.getByName(interfaceName);

                // Get the IP and the mask from the NetworkInterface
                List<InterfaceAddress> cardAddresses = networkCard.getInterfaceAddresses();
                for (InterfaceAddress ifAddr : cardAddresses) {
                    if (ifAddr.getAddress() instanceof Inet4Address) {
                        ipAddress = ifAddr.getAddress().getHostAddress();
                        networkMask = Integer.toString(ifAddr.getNetworkPrefixLength());
                    }
                }

            }
            case 2 -> {
                // Get the IP and mask
                ipAddress = args[0];
                networkMask = args[1];

                // Get the interface name from the assigned IP
                networkCard = NetworkInterface.getByInetAddress(InetAddress.getByName(ipAddress));
                interfaceName = networkCard.getName();
            }
            default -> {
                // In order to find the right interface name... run the app with no args
                NetworkInterface.networkInterfaces().forEach((t) -> {
                    try {
                        if (t.isUp())
                            System.out.println(t.getName() + " --> " + t.getDisplayName() + " : " + t.inetAddresse
                    } catch (SocketException ex) {
                        Logger.getLogger(U4S4_HostSeeker.class.getName()).log(Level.SEVERE, null, ex);
```

```
                }
            });

            System.err.println("Sysntax error. Usage: U4S4_HostSeeker IP maskLength");
            System.err.println("Sysntax error. Usage: U4S4_HostSeeker interfaceName");
            throw new AssertionError();
        }
    }

}

private void processNetwork() throws IOException {
    String baseAddress;
    String[] addressParts = ipAddress.split("\\.");
    switch (networkMask) {
        case "24" -> {
            baseAddress = addressParts[0] + "." + addressParts[1] + "." + addressParts[2] + ".";
            checkClassCNetwork(baseAddress);

        }
        case "16" -> {
            baseAddress = addressParts[0] + "." + addressParts[1] + ".";
            checkClassBNetwork(baseAddress);
        }
        case "8" -> {
            baseAddress = addressParts[0] + ".";
            checkClassANetwork(baseAddress);

        }
        default ->
            throw new AssertionError();
    }
}

// Check for /24 networks
private void checkClassCNetwork(String baseAddress) throws UnknownHostException, IOException {
    System.out.println("Checking network " + baseAddress + "0...");
    for (int i = 1; i < 255; i++) {
        InetAddress tempIP = InetAddress.getByName(baseAddress + i);
        if (tempIP.isReachable(timeout)) {
            addReachableIp(tempIP.getHostAddress());
        }
    }
}

// Check for /16 networks
private void checkClassBNetwork(String baseAddress) throws UnknownHostException, IOException {
    for (int i = 0; i <= 255; i++) {
        System.out.println("Checking next 256 class C adresses: " + baseAddress + i + ".xxx");
        checkClassCNetwork(baseAddress + i + ".");
    }
}

// Check for /8 networks
private void checkClassANetwork(String baseAddress) throws UnknownHostException, IOException {
    for (int i = 0; i <= 255; i++) {
```

```
            System.out.println("Checking next 256 class B adresses: " + baseAddress + i + ".xxx");
            checkClassBNetwork(baseAddress + i + ".");
        }
    }

    private /*synchronized*/ void addReachableIp(String ip) {
        reachableIps.add(ip);
    }

    private void listResults() {
        System.out.println("Reachable IPs:");
        for (String ip : reachableIps) {
            System.out.println("--> " + ip);
        }
    }
}
```

java.net.URL

URL es un acrónimo de Localizador de recursos uniforme. Un recurso puede ser cualquier cosa, desde un simple archivo de texto hasta cualquier otro como imágenes, directorio de archivos, etc.

> java.net.URL specification

Es un puntero para localizar recursos en www (World Wide Web), por ejemplo:

> http://psp2dam.github.io/psp_pages/

La URL tiene las siguientes partes:

- **Protocolo**: en este caso el protocolo es HTTP, puede ser HTTPS en algunos casos
- **Nombre de host o IP**: el nombre de host representa la dirección de la máquina en la que se encuentra el recurso, en este caso, www.example.com
- **Número de puerto**: es un atributo opcional. Si no se especifica, devuelve -1. En el caso anterior, el número de puerto es 80. Si no se indica se usa el puerto usado por defecto por el protocolo indicado en el primer campo.
- **Nombre del recurso**: es el nombre de un recurso ubicado en el servidor dado que queremos ver (la carpeta /psp_pages). Dependiendo de la configuración del servidor, el nombre del archivo puede tener un valor por defecto. En el ejemplo sólo se ha indicado una ruta, por lo que se intentará devolver el archivo `index.html` si se encuentra en esa carpeta.

Constructor	Descripción
URL(String spec)	Este constructor crea un objeto de clase URL a partir de una representación de string dada

Constructor	Descripción
URL(String protocol, String host, int port, String file)	Este constructor crea un objeto de URL a partir del protocolo, host, número de puerto y archivo especificados.
URL(String protocol, String host, String file)	Este constructor crea un objeto de URL a partir del protocolo, el servidor y la ruta/archivo especificados.
URL(URL context, String spec)	Este constructor crea una instancia de una URL analizando el src dado con el controlador especificado dentro de un contexto dado. Se usa cuando tenemos rutas relativas a una URL

Estos son los métodos mas importantes y utilizados de la clase de URL:

Método	Descripción
public String getProtocol()	Este método obtiene el nombre de protocolo de esta URL.
public String getHost()	Este método obtiene el nombre de host de esta URL, si corresponde.
public String getPort()	Este método obtiene el número de puerto de esta URL.
public String getFile()	Este método obtiene la parte de la ruta de esta URL.
public String getAuthority()	Este método obtiene la parte de autoridad de esta URL.
public String toString()	Este método construye una representación de string de esta URL.
public String getQuery()	Este método obtiene la parte de consulta de esta URL.
public String getDefaultPort()	Este método obtiene el número de puerto predeterminado del protocolo asociado con esta URL.
public URLConnection openConnection()	Este método devuelve una instancia de URLConnection que representa una conexión al objeto remoto al que hace referencia la URL.
public InputStream openStream()	Este método abre una conexión a esta URL y devuelve un InputStream para leer desde esa conexión.
public boolean equals(Object obj)	Este método compara la igualdad de esta URL con otro objeto.
public Object getContent()	Este método obtiene el contenido de esta URL.
public String getRef()	Este método obtiene el ancla (también conocido como la «referencia») de esta URL.
public URI toURI()	Este método devuelve un URI equivalente a esta URL.

```
//URLDemo.java
public static void main(String[] args) throws MalformedURLException{

    URL url=new URL("http://psp2dam.github.io/psp_pages");

    System.out.println("Protocol: "+url.getProtocol());
    System.out.println("Host Name: "+url.getHost());
    System.out.println("Port Number: "+url.getPort());
    System.out.println("File Name: "+url.getFile());
}
```

Vamos con otro ejemplo más completo de uso de los métodos de URL.

```
//URLDemo.java
public static void main(String[] args){
    URL url=new URL("https://www.google.com/search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8");

    System.out.println("Protocol: "+url.getProtocol());
    System.out.println("Host Name: "+url.getHost());
    System.out.println("Port Number: "+url.getPort());
    System.out.println("Default Port Number: "+url.getDefaultPort());
    System.out.println("Query String: "+url.getQuery());
    System.out.println("Path: "+url.getPath());
    System.out.println("File: "+url.getFile());
}
```

```
Protocol: https
Host Name: www.google.com
Port Number: -1
Default Port Number: 443
Query String: q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8
Path: /search
File: /search?q=javatpoint&oq=javatpoint&sourceid=chrome&ie=UTF-8
```

java.net.URLConnection

La clase URLConnection se utiliza para dos propósitos diferentes pero relacionados.

- En primer lugar, proporciona más control sobre la interacción con un servidor (especialmente un servidor HTTP) que la clase URL.
- En segundo lugar, con una URLConnection podemos verificar el encabezado enviado por el servidor y responder en consecuencia, podemos configurar los campos de encabezado utilizados en las requests de los clientes. También podemos descargar archivos binarios usando URLConnection.

La clase URLConnection representa un enlace entre la URL y la aplicación. Puede usarse para leer y escribir datos en el recurso señalado por la URL.

> java.net.URLConnection specification

La clase URLConnection en Java es una clase abstracta que representa una conexión de un recurso según lo especificado por la URL correspondiente. Tiene dos subclases `HttpURLConnection` y `JarURLConnection` que se encargan de hacer la conexión entre el programa cliente y el recurso indicado en URL.

La clase URLConnection proporciona una gran cantidad de métodos. Podemos mostrar el contenido de una página web usando el método `getInputStream()` , de un modo similar a como lo hacíamos con los procesos.

Método	**Descripción**
void connect()	Este método se utiliza para establecer una conexión con el recurso especificado por la URL, si dicha conexión aún no se ha establecido.
Object getContent()	Recupera el contenido de esta conexión URL.
String getContentEncoding()	Devuelve el valor del campo de encabezado de codificación de contenido.
int getContentLength()	Devuelve el valor del campo de encabezado de longitud del contenido.
long getContentLengthLong()	Devuelve el valor del campo de encabezado de longitud del contenido como long.
String getContentType()	Devuelve el valor del campo de encabezado de tipo de contenido.
long getDate()	Devuelve el valor del campo de encabezado de fecha.
boolean getDoInput()	Devuelve el valor del indicador doInput de esta URLConnection.
boolean getDoInput()	Devuelve el valor del indicador doInput de esta URLConnection.
String getHeaderField(int n)	obtiene el valor del enésimo campo de encabezado.
String getHeaderField(String name)	Devuelve el valor del campo de encabezado con nombre.
String getHeaderFieldKey(int n)	obtiene el valor del enésimo campo de encabezado.
Map<String, List<String>> getHeaderFields()	Devuelve un mapa no modificable de los campos de encabezado.
long getIfModifiedSince()	Devuelve el valor del campo ifModifiedSince de este objeto.
InputStream getInputStream()	Devuelve un flujo de entrada que lee de esta conexión abierta
long getLastModified()	Devuelve el valor del campo de encabezado modificado por última vez.
OutputStream getOutputStream()	Devuelve un flujo de salida que escribe en esta conexión.
URL getURL()	Devuelve el valor del campo URL de este URLConnection.
void setDoInput(boolean doinput)	Establece el valor del campo doInput para esta URLConnection en el valor especificado.

Método	Descripción
void setDoOutput(boolean dooutput)	Establece el valor del campo doOutput para esta URLConnection en el valor especificado.

Cómo obtener un objeto de tipo URLConnection

El método openConnection() de la clase URL devuelve un objeto de tipo URLConnection.

```
// URLConnectionExample
public static void main(String[] args) throws MalformedURLException, IOException{

    // Creating an object of URL class

    // Custom input URL is passed as an argument
    URL u = new URL("www.google.com");

    // Creating an object of URLConnection class to
    // communicate between application and URL
11  URLConnection urlconnect = u.openConnection();

    // Creating an object of InputStream class
    // for our application streams to be read
    InputStream stream
        = urlconnect.getInputStream();

    BufferedReader in =
        new BufferedReader(
            new InputStreamReader(stream));
    // Till the time URL is being read
    String line;
    while ((line = in.readLine()) != null) {

        // Continue printing the stream
        System.out.println(line);
    }
}
```

MalformedURLException

Si pruebas el código anterior, obtendrás una excepción de tipo `MalformedURLException`.
¿Qué deberías cambiar para que funcione correctamente?

Descargar imágenes (U4S7_ImagesDownloader)

Crea una nueva aplicación que descargue imágenes de una URL.

La URL de la imagen se debe pasar como argumento a la aplicación. La imagen descargada debe guardarse en una carpeta images, en la raíz del proyecto, con el mismo nombre que tuviese el recurso online.

Código del ejemplo

```
// Code not visible yet
```