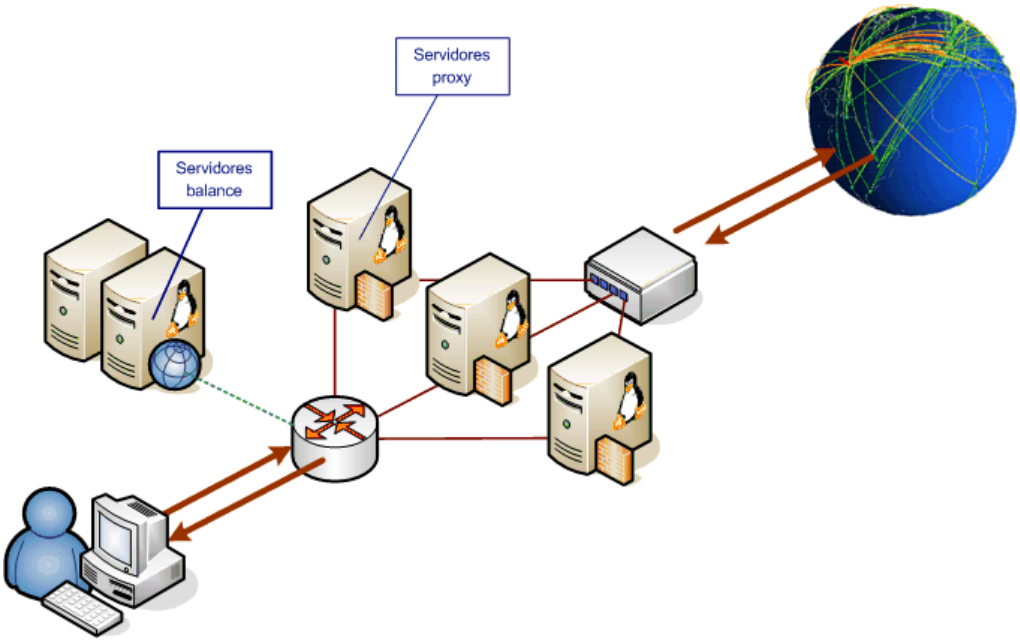
UD04: Network services coding



1. Introduction

Computer networks (and other devices) are an incredible source of opportunities to develop applications and services. Creating programs that run distributed among several computers, obtaining information from web pages and services, transferring files or sending emails are some of the most interesting examples.

The technologies and protocols on which the Internet is based are free to use, which means that they can be used from any programming language. Furthermore, by using the appropriate classes and libraries, programming applications that make use of these technologies is relatively simple and fast. Consequently, the development of applications that take advantage of the potential of networks is a highly attractive task.

This unit explains the programming techniques of the most popular services based on the use of networks based on Internet technologies, as well as the most convenient libraries and classes to carry them out.

2. Standard network communication protocols at the application level

The OSI and TCP/IP network models are structured as a succession of layers responsible for all activity related to communication between devices over a network. Each layer has a different level of abstraction, depending on the task it has to perform.

The lower layers are much more technical and dry to program than the upper ones, since they are in charge of much more specific details related to communication. The programs of the lower layers are found in the drivers of the network cards and in the operating systems.

The upper layers are the ones used in building the applications that users use. In fact, the upper layer is called "application" both in the theoretical OSI model and in the TCP/IP model used on the Internet.

Therefore, the application layer of the TCP/IP network model contains the applications and services that the user can use. It is the layer closest to people and on which everyday applications will be developed.

Within the application layer are several protocols, each of which has a specific role. The number of protocols is extensive, so only the most relevant ones are presented in this unit.

2.1. HTTP and HTTPS

The Hypertext Transfer Protocol (HTTP or Hypertext Transfer Protocol) is used to transmit documents over the Internet. HTTP is the fundamental protocol of the web, since it allows the transfer of requests and resources exchanged by clients and servers. It is based on the TCP protocol for its operation. For its part, the Hypertext Transfer Protocol Secure (HTTPS or Hypertext Transfer Protocol Secure) is the secure version of the HTTP protocol, since it encrypts all information exchanged between client and server.

It is not necessary to know the HTTP protocol in depth to develop applications that carry out network communications, but there are some technical data that should be kept in mind.

One of the technical aspects that it is necessary to know about HTTP is the one referring to the types of requests that can be made. It is known as method and they are those shown in the following table:

Method	Description
`GET`	Requests the recovery of a resource. An HTTP request using this method retrieves an entity hosted on the server.
`POST`	This method is used to create a hosted entity on the server.
`PUT`	This method is used to create or update an entity hosted on the server if it already existed.
`DELETE`	Deletes an entity hosted on the server.

Another of the HTTP elements that is essential to know is the reference to the response codes. When an HTTP request is made to a server, it generates a response that is identified with a code. This code provides information on the result of the request processing, letting you know if it has been processed correctly or if some problem has arisen. These codes are made up of three digits, the first being the one that globally determines the type of response. These groups are shown in the following table.

Return codes	Description
100-199	Informative answer.
200-299	Success.
300-399	redirect.
400-499	Client error.
500-599	Server error.

Of the codes contained in these groups, some of the most frequent are 200 (the request has been successful) and 404 (the requested resource has not been found). In applications that make HTTP requests, the code returned by them must be evaluated to find out if they have been processed correctly or if there has been a problem.

By default HTTP uses port 80 while HTTPS uses 443.

2.2. FTP

The File Transfer Protocol allows the transfer of files of all kinds between devices. It is based on a client-server architecture and uses the TCP protocol.

FTP uses port 21 by default.

2.3. SMTP

The Simple Mail Transfer Protocol (SMTP or Simple Mail Transfer Protocol) is used for sending emails. It uses the MIME specification (Multipurpose Internet Mail Extensions or multipurpose Internet mail extensions) that allows the exchange of all types of files over the Internet.

By default it uses port 25.

2.4. IMAP

The Internet Message Access Protocol (IMAP) is used to receive emails. IMAP stores emails on the server so they can be read from different devices.

It uses port 143 for unencrypted connections and port 993 for encrypted ones.

2.5. POP3

The Post Office Protocol (POP3 or Post Office Protocol) is, like IMAP, the one used to receive emails. POP3 downloads the emails and removes them from the server, so once they are consulted they cannot be accessed from devices other than the first access.

It uses port 110 for unencrypted connections and port 995 for encrypted ones.

2.6. DNS

The Domain Name System (DNS or Domain Name System) allows the association of the names of the devices connected to the network with their IP addresses. It is the protocol that allows a URL or an email address to be linked to the IP address where the related service is located.

The default port used by this protocol is 53.

2.7. TELNET

This protocol allows access through a terminal (without graphics) to a remote computer. It does not encrypt the information sent and received, so it is considered insecure.

Telnet uses port 23.

2.8. SSH

SSH (Secure Shell) is a protocol with a similar objective to Telnet, since it allows remote access to a computer through a terminal. In this case, the communication is encrypted by what is considered a secure protocol.

By default it uses port 22.

2.9. LDAP

The Lightweight Directory Access Protocol (LDAP) provides a hierarchical, ordered and distributed structure of information, as well as the tools for accessing it. It is usually used to store information regarding system access (credentials and permissions).

The port used by default is 389.

2.10. NFS

The Network File System (NFS or Network File System) allows you to distribute files on a network and access them from any node on it.

It uses port 2049.

2.11. SNMP

The Simple Network Management Protocol (SNMP or Simple Network Management Protocol) provides the ability to exchange information between network devices that are part of its infrastructure (routers, switches, etc.), having a vocation clearly oriented towards administration of the networks

It uses port 161.

2.12. DHCP

The Dynamic Host Configuration Protocol (DHCP or Dynamic Host Configuration Protocol) is responsible for the dynamic assignment of IP addresses and configuration parameters to devices that are part of a network.

When the addresses are not fixed in a network, the DHCP service is responsible for assigning each device an IP address from a list of available addresses, allowing dynamic management of these addresses.

This protocol uses port 67 by default.

2.13. SMB and CIFS

These two protocols allow sharing resources on a network, such as files or printers. SMB (Server Message Block) and CIFS (Common Internet File System) are closely related to each other, as CIFS is an implementation of SMB created by Microsoft.

Currently, the recommendation is to use SMB 2 and SMB 3 as implementations of this protocol.

SMB and CIFS use ports 139 and 445 by default.

3. Classes and libraries for creating network services

Virtually all modern programming languages have libraries for programming applications that make use of the network. It must be taken into account that, in the absence of these libraries, the programming would be at a fairly low level, requiring the use of sockets, byte transfer and exhaustive knowledge of protocols.

Java natively has a good set of libraries and classes that support the development of this type of application. In addition, other developers provide libraries and classes that improve or complement the language's own.

3.1. Standard Java classes

The classes natively available in Java provide all the functionality needed to implement networked applications. The most relevant are presented in this section.

3.1.1. `java.net.URL`

Represents a URL (Uniform Resource Locator), a reference to a web resource.

The following code shows the construction of an object of this class.

```
1 URL url = new URL("http://www.martinezpenya.es");
```

If the web address given in the constructor is not correctly formed, it will throw the `MalformedURLException` exception.
The most important method is shown in the following table:

Method	Description
`openConnection`	Provides a connection (`URLConnection` object) from the resource represented in the URL.

3.1.2. `java.net.URLConnection`

This abstract class is the superclass of all classes that represent a communication link between the application and a URL.

Instances of this class allow reading and writing to the resource referenced by the URL.

`HttpURLConnection` is the most relevant subclass.

The most important methods are shown in Table format below:

Method	Description
`getByName`	Static method that provides the IP address of a host from its name.
`getLocalHost`	Static method that provides the IP address of the local host.
`getHostAddress`	Provides the IP address of the host.
`getHostName`	Provides the alias of the host.
`getAddress`	Returns the host's IP address as a byte array.
`getCanonicalHostName`	Provides the host name.
`getInputStream`	Provides a read stream.
`getOutputStream`	Provides a write stream.
`setRequestProperty`	Assigns the value to a property.

The following sample code establishes a connection to a web resource and reads its content:

```
URL url = new URL("http://www.martinezpenya.es”);
URLConnection conexionURL = url.openConnection();
InputStream is = conexionURL.getInputStream();
int c;
while ((c=is.read())!=-1) {
    system.out .print ((char) c);
}
is.close();
```

Class available since Java version 1.0.

3.1.3. `java.net.HttpURLConnection`

This class provides the mechanisms to manage an HTTP connection.

The following table shows the main methods. It should be taken into account that inheriting from `URLConnection` also has its same methods.

Method	Description.
`disconnect`	Disconnect the connection.
`getResponseCode`	Provides the HTTP return code sent by the server.
`setRequestMethod`	Provides the request method.

It also contains the constants that represent the status codes of the HTTP protocol, such as `HttpURLConnection.HTTP_OK` which has the integer value 200 and means that the request was successful (OK).

Class available since version 1.1 of Java.

3.1.4. `java.net.http.HttpClient`

This abstract class allows you to make HTTP requests and get their responses. The instances must be created through a builder or instantiation object.

Some of the most relevant methods are those shown in the following table:

Method	Description
`newBuilder`	Create a builder (interface object `HttpClient.Builder`
`send`	Sends the HTTP request and returns an instance of `HttpResponse`. It receives as a parameter, in addition to the request, an object of the `HttpResponse.BodyHandlers` class, in charge of managing the content of the request response.

The methods of `HttpClient.Builder` are listed below:1

Method	Description
`build`	Provides the `HttpClient` object with the provided configuration.
`followsRedirect`	It provides mechanisms to determine how the request should behave against redirects from the server.
`version`	Allows you to specify the version of the HTTP protocol.

Class available since version 11 of Java.

3.1.5. `java.net.http.HttpRequest`

Abstract class that represents an HTTP request. The instances are configured and created through a constructor or builder. This constructor is obtained through the newBuilder static method of the class itself, which will be indicated the HTTP method to use, the request parameters or the waiting timeout, among other configuration parameters.

The main method is:

Method	Description
`newBuilder`	Static method that creates a builder (interface object `HttpRequest.Builder`).

For its part, the `HttpRequest.Builder` interface provides the following methods:

Method	Description
`build`	Provides the `HttpClient` object with the provided configuration.
`DELETE`	Assign the `DELETE` method to the builder.
`GET`	Assign the `GET` method to the builder.
`header`	Allows you to add a parameter-value pair to the request.
`headers`	Allows adding parameter-value pairs to the request.
`POST`	Assign the `POST` method to the builder.
`PUT`	Assign the `PUT` method to the builder.
`setHeader`	Allows you to assign a key-value pair to the request.
`timeout`	Allows you to determine a time limit (`timeout`) for the request.
`uri`	Assign the `URI` to the request.
`version`	Allows you to specify the version of the HTTP protocol.

Class available since version 11 of Java.

3.1.6. `java.net.http.HttpResponse`

Interface that represents the response of an HTTP request. Instances of this interface are not created directly, but are provided by the send method of the HttpClient class.

The most important method is:

Method	Description
`statusCode`	Provides the HTTP request status code.

The `HttpResponse` class allows you to get instances of the `HttpResponse.BodyHandler` functional interface. These instances are used as parameters in the call to the `send` method of `HttpClient` and determine the way in which the response body of the HTTP request is to be processed.

It has several static methods to obtain the `BodyHandler` instances, the most relevant being:

Method	Description
`ofByteArray`	static method. Returns an object of type `BodyHandler<byte[]>`.
`ofFile`	static method. Returns an object of type `BodyHandler<Path>`.
`ofInputStream`	static method. Returns an object of type `BodyHandler<InputStream>`.
`ofString`	static method. Returns an object of type `BodyHandler<String>`.

The `BodyHandler` interface only has a single method, `apply`, which is called automatically when used. Depending on the type of class used, this method will perform one or another action.

For example, upon execution of the following code, a call to `response.body()` returns a byte array:

```
1 HttpResponse<byte[]> response = httpClient.send(request,
  HttpResponse.BodyHandlers.ofByteArray());
```

En cambio, la misma llamada al método `body()` sobre el objeto response obtenido en la siguiente ejecución, proporciona un `String`:

```
1 HttpResponse<String> response = httpClient.send(request,
  HttpResponse.BodyHandlers.ofString);
```

Finally, the execution of the following statement stores the content of the response body in the file indicated in the `path` variable.

```
1 HttpResponse<Path> response = httpClient.send(request,
  HttpResponse.BodyHandlers.ofFile(Path.of(path)));
```

Class available since version 11 of Java.

3.1.7. `JavaMail`

The `JavaMail` API provides a protocol and platform independent environment for building messaging applications via email.

It is a package included in the enterprise version of Java (Java EE or Java Enterprise Edition) that can optionally be included in projects developed with the standard version (Java SE or Java Standard Edition).

Some of the most important classes of this library are:

Class	Description
`com.sun.mail.imap.IMAPFolder`	Represents an IMAP message folder. Inherits from `javax.mail.Folder`.
`com.sun.mail.pop3.POP3Folder`	Represents a POP3 message folder. Inherits from `javax.mail.Folder`.
`javax.mail.Address`	Abstract class that represents an email address.
`javax.mail.Folder`	Abstract class that represents a message folder.
`javax.mail.Message`	Abstract class that represents an email message.
`javax.mail.Message.Recipienttype`	Inner static class. Defines the types of recipients allowed. The types defined are TO, CC, and BCC.
`javax.mail.Multipart`	Abstract class representing a container that supports multiple message body parts (body).
`javax.mail.Session`	Represents an email session.
`javax.mail.Store`	Abstract class representing a message store and its access protocol. Instances of this class are obtained by calling the `getStore` method of a `Session` object.
`javax.mail.Transport`	Abstract class that represents the transport method of the messages. Instances of this class are obtained by calling the `getTransport` method of a `Session` object.
`javax.mail.internet.InternetAddress`	Represents an Internet email address using the RFC822 standard. Inherits from `javax.mail.Address`.
`javax.mail.internet.MimeBodyPart`	Represents a part of the body (`body`) of a MIME message.
`javax.mail.internet.MimeMessage`	Represents an email message that uses the MIME convention. Inherits from the `javax.mail.Message` class.
`javax.mail.internet.MimeMultipart`	Implementation of `javax.mail.Multipart` that uses the MIME convention

3.2. `Apache Commons` libraries

Apache Commons is a set of open source Java libraries created under the umbrella of the Apache Software Foundation, a non-profit software development organization.

The `Apache Commons` libraries cover various aspects of software development through components dedicated to topics as diverse as trace management (logs), mathematical and statistical operations, or random number generation, to name a few examples.

Regarding the generation of network services, the most relevant libraries are `Commons Email` for sending and receiving emails and `Apache Commons Net` with implementations of internet protocols.

These libraries provide alternative implementations to those available in the Java language.

4. Communication via HTTP

The generation of services based on the HTTP protocol has two perspectives, corresponding to the client and the server. This book addresses the client perspective, learning to develop Java applications that are capable of making HTTP requests to obtain information provided by web sites or services. The other perspective provides the ability to develop web applications and services, but is left out
the scope of this book.

Due to issues related to licenses and the evolution of Java versions, there are two models for making HTTP requests: the one used up to Java version 1.8 and the one incorporated from version 11. In this unit, we will Present both options.

4.1. HTTP requests based on `HttpUrlConnection`

Until Java version 1.8, the `HttpURLConnection` class from the `java.net` package has been the basis for programming applications capable of accessing web resources.

The steps to make an HTTP request without parameters with this class are the following:

- Creation of URLs.
- Opening of the connection.
- Connection configuration:
    - HTTP method.
    - Type of content.
    - Coding system.
    - User agent to use.
- Obtaining and evaluating the HTTP response code. If the code is 200:
    - Obtaining the InputStream object for reading (if applicable).
        - Stream reading.
    - Obtaining the OutputStream object for writing (if applicable).
        - Writing in the stream.
    - Disconnection.

Look at Example1 and Example2

4.2. HTTP requests based on `java.net.http`

Starting with Java version 11, the `java.net.http` package was introduced to provide a more powerful, easier, and up-to-date alternative to making HTTP requests (it supports HTTP/1.1, HTTP/2, and WebSockets).

> WebSockets allow event-based, bidirectional communication to be established between a web server and a browser.

Three classes are mainly used to carry out conventional HTTP requests:

- `HttpClient`
- `HttpRequest`
- `HttpResponse`

To make an HTTP request, the following steps must be performed:

- Create the `HttpClient` object, indicating the protocol version, as well as other optional data such as the behavior in case of server redirections.
- Create the `HttpRequest` object, indicating the URI and the request header parameters.
- Make the request through the send method of the `HttpClient` and assign the response of the request to an `HttpResponse` object.
- Process the response.

Look at the Example3

5. File transfer via FTP

In Java, natively, it is possible to carry out file transfers using this protocol, but it is extremely dry.

The Apache Commons Net library provides classes and utilities to perform any operation on an FTP or FTPS server from a Java client.

This library can be downloaded from the apache.org website through the following link: https://commons.apache.org/proper/commons-net/

The main classes are in the `org.apache.commons.net.ftp` package and are shown below.

Class	Description
`FTP`	It provides the functionality necessary to implement an FTP client. It includes constants to indicate the type of files to be transmitted and their configuration.
`FTPClient`	FTP subclass, encapsulates the functionality needed to upload and download files via the FTP protocol.
`FTPSClient`	Subclass of FTPClient, allows to use the FTP protocol over SSL.
`FTPFile`	Represents information about files stored on the server.
`FTPReply`	Stores the constants that represent the return codes of the FTP protocol.

The `FTPClient` class is the one that provides the communication methods with the server, allowing all the operations that the FTP protocol supports. The most common methods are:

Method	Description
`connect`	Allows you to connect to a server based on its hostname port (method inherited from org.apache.commons.net.SocketClient). andsu
`changeToParentDirectory`	Changes the server's working directory to the parent directory of the current one.
`changeWorkingDirectory`	Change the working directory of the server.
`deleteFile`	Delete a file on the server.
`disconnect`	Close the connection to the FTP server.
`getReplyCode`	Provides the response code of an FTP request (method inherited from `org.apache.commons.net.ftp.FTP`).
`listDirectories`	Provides the existing directories in the server's working directory.
`listFiles`	Provides existing files in the server's working directory.
`login`	Allows access to the server using a user account.
`logout`	Disconnects the account of the validated user.
`makeDirectory`	Create a directory on the server.
`rename`	Renames a file on the server.
`retrieveFile`	Download a file from the server to the client.
`setFileType`	Allows you to indicate the type of file to be transferred.
`storeFile`	Upload a file from the client to the server.

FTP user accounts can have different permissions, so the methods of the FTPClient class will work depending on whether these permissions are correctly configured.

Check the Example4

6. Sending and receiving emails

Sending and receiving email from native Java classes is possible but extremely dry and time consuming. Fortunately, there are alternatives that provide simpler and more compact mechanisms.

The JavaMail framework has the necessary classes to be able to program the most common actions that a computer system could perform in relation to emails, such as sending and receiving messages with and without attachments.

You can use `JavaMail` by downloading and adding to the project the libraries contained in the `javax.mail.jar` and `javax.activation-1.2.0.jar` files (Not to be confused with `javax.activation-api-1.2.0.jar`).

The process of sending emails composed only of texts consists of the following steps:

- Creation of the session, indicating the URL of the SMTP server, the port, if it uses SSL and if authentication is required.
- Creation of the message (Message object). This object includes the sender's email address, the recipient's email address, the subject and the text of the message.
- Establishment of the connection (creation of the Transport object), indicating the transport system.
- Sending the message.
- Closing the connection.

If the mail has attached files, the creation of the message is divided into several stages. The complete process is as follows:

- Creation of the session, indicating the URL of the SMTP server, the port, if it uses SSL and if authentication is required.
- Creation of the message (Message object). This object includes the sender's email address, the recipient's email address, and the subject. In addition, the creation of the message implies:
    - Creation of the `BodyPart` instance that contains the text of the message.
    - Creation of the `MimeBodyPart` instance that contains the message attachment.
    - Creation of the `Multipart` instance, which groups the `BodyPart` object and the `MimeBodyPart` object.
    - Inclusion of the `Multipart` object in the message.
- Establishment of the connection (creation of the Transport object), indicating the transport system.
- Sending the message.
- Closing the connection.

Check the Example5

For its part, reading emails stored on an IMAP server consists of the following steps:

- Creation of the IMAP session (Session), indicating the protocol, the host name, the port, if it uses SSL and the associated trusted server.
- Configuration and obtaining of the warehouse (Store).
- Obtaining the connection through the store, indicating the account identifier and password.
- Obtaining the folder to be read.
- Opening of the folder.
- Getting the messages
- Message processing

Check the Example6

7. Distributed programming

Distributed programming is a programming paradigm consisting of distributing a software system among a set of computers connected in a network. The advantage of this computing system is that usually the computing power that can be obtained through the joint use of a group of computers has a much lower cost than that of a single computer with the same computing power.
calculation.

In a distributed system, the software system is divided into components and these are distributed and executed on different computers, unifying the results obtained later. It is easy to sense that not all problems are likely to be executed in a distributed environment and obtain obvious improvements. However, in many cases, distributed programming is the best alternative to carry out complex computational processes.

The main elements that are part of a distributed system are:

- Nodes. Each of the computers that are part of the network.
- Software components. The software elements that implement the functionality of the system, mainly classes.
- Remote registration of objects. Network element that knows the location of the components in the nodes.
- Network and protocols. Physical and logical infrastructure necessary to communicate the different network participants.
- A Remote interface. It is the interface shared by the client and the server, although the implementation is on the server. The client, for his part, will obtain a remote instance of said interface and perform the operations on it.
- Stubs. It is the projection of the remote object on the client. The client invokes the remote methods on the stub, and the stub propagates the calls to the equivalent remote implementation, or skeleton.
- Skeleton. It is the remote object that receives the calls from the stub and causes the functionality to be executed on the server side.

In a distributed system, all nodes in the network can be clients and servers. There is no single server, since the system is partitioned (distributed) among the nodes. Each network element may have and run one or more parts of the system, acting as a single whole. The components are distributed among the nodes that form it, obtaining the following advantages compared to a conventional system:

- **Powerful**. The sum of the processors of the nodes makes the computing capacity of a distributed system potentially unlimited.
- **Scalable**. Since by definition the system is distributed among several computers, the number of these can grow in number and capacity.
  Fault tolerant. The components of the system are distributed among the various nodes, but there is no predetermined node-component relationship. If a node goes down, the components it provided can be switched to being provided by other nodes.

- **Efficient**. Several computers of moderate capacity and cost can provide the same computing capacity as one computer of high capacity and cost with a lower investment.
- **Transparent**. Given the architecture of distributed systems, the nodes do not have information about the other nodes. The system itself acts as an intermediary between them, so changes in the network, either by adding or removing nodes,
  they do not affect your system.

The concrete operation of a distributed system that uses the paradigm of object orientation is the following. The instances are distributed among the nodes of the distributed system, each one executing on its own processor, in a way playing the role of servers. The application that needs these instances to run accesses an intermediary service that provides remote references to the objects.

From a programmatic point of view, the invocation of the methods on the remote objects is done locally, but the execution itself is done on the computer that has the instance.

In a more schematic way, the steps to build and use a distributed system are the following:

- Server elements:
  - Creation of the objects and Installation in the nodes of the network.
  - Registration of objects in the remote registry of objects.
- Customer elements:
  - Access to the registry of objects to obtain the remote references to the objects.
  - Invocation of the methods provided by the remote objects.

Realize that remote references are projections of the server object onto the client machine on which they are being used. It can be summarized that the object is remote, but the use is local.

Java has its own technology for developing distributed applications, called Java Remote Method Invocation or RMI.
This technology provides the necessary tools to create the various components that participate in an application developed on this architecture. Of all the components provided, the most relevant are the following:

- The `java.rmi.Remote` interface that allows you to define methods that can be executed on a non-local Java virtual machine. Methods defined in this interface must throw the `java.rmi.RemoteException` exception. Used on client and server.
- The `java.rmi.registry.Registry` interface allows the registry of remote objects to store and retrieve references to them. Used on client and server.
- The final class `java.rmi.registry.LocateRegistry` used to get the reference to the remote registry objects (Registry). Used on client and server.
- The `java.rmi.server.UnicastRemoteObject` class used for remote object export and stub generation. Used on client and server.
- `rmiregistry` program, responsible for starting the registry of remote objects on a given host and port. It is distributed together with the Java JDK.

Check the Example7

8. Examples of Unit 04

8.1. Example 1

HTTP Request with `HttpUrlConnection` (Java 8)

The website of the dictionary of the RAE (Royal Spanish Academy) allows you to consult the meaning of the words of the Spanish language. It has a simple form that facilitates the introduction of a word to carry out the search. This word is concatenated to the URL to request the corresponding resource.

For example, if the word software is entered in the search form, the resource https://dle.rae.es/software is requested.

Therefore, the request is of the GET type and has no parameters.

In this example we are going to create a Java program that makes requests to the RAE website and that stores the HTML code obtained in a file.

We must highlight the need to encode the searched word using the `encode` method of the `URLEncoder` class so that characters such as the letter `ñ`, blank spaces or vowels with tilde have the format accepted by the protocol.

```
1  package UD04.Ejemplo01;
2
3  import java.io.IOException;
4  import java.io.InputStreamReader;
5  import java.io.Reader;
6  import java.net.HttpURLConnection;
7  import java.net.URL;
8  import java.net.URLEncoder;
9  import java.nio.charset.StandardCharsets;
10 import java.nio.file.Files;
11 import java.nio.file.Path;
12 import java.nio.file.Paths;
13
14 public class RAE {
15
16     public static StringBuilder htmlDownload(String address) throws Exception {
17         StringBuilder answer = new StringBuilder();
18         URL url = new URL(address);
19         HttpURLConnection connection = (HttpURLConnection) url.openConnection();
20         connection.setRequestMethod("GET");
21         connection.setRequestProperty("Content-Type", "text/plain");
22
23         connection.setRequestProperty("charset", "utf-8");
24         connection.setRequestProperty("User-Agent", "Mozilla/5.0");
25         int state = connection.getResponseCode();
26         Reader streamReader = null;
27         if (state == HttpURLConnection.HTTP_OK) {
28             streamReader = new InputStreamReader(connection.getInputStream());
```

```
            int character;
            while ((character = streamReader.read()) != -1) {
                answer.append((char) character);
            }
        } else {
            throw new Exception("HTTP Error: " + state);
        }
        connection.disconnect();
        return answer;
    }

    public static void writeFile(String strPath, String content) throws IOException {
        Path path = Paths.get(strPath);
        byte[] strToBytes = content.getBytes();
        Files.write(path, strToBytes);
    }

    public static void main(String[] args) {
        try {
            String scheme = "https://";
            String server = "dle.rae.es/";
            String resource = URLEncoder.encode("Tiburón",
                    StandardCharsets.UTF_8.name());
            String address = scheme + server + resource;
            StringBuilder result = htmlDownload(address);
            RAE.writeFile("src/UD04/Ejemplo01/tiburon.html",
                    result.toString());
            System.out.println("Download completed");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

8.2. Example2

HTTP request with parameters with `HttpUrlConnection` (Java 8)

IMDb (Internet Movie Database or Internet Movie Database) is a website that contains an infinite number of information related to film and television, including movies, series, actors, directors, etc.

This website provides a search form in which you can enter any word or words to search for movies, actors or companies that contain those words.

When entering the words, the form generates a GET type HTTP request that includes a parameter with the name q and the value of the words entered. For example, if the words star wars are entered, the request created is https://www.imdb.com/find?q=star+wars.

In this example, a Java program will be created that, given a word or words, makes an HTTP request to the IMDb website and stores the result in an HTML file.

It is exactly the same as Example1 except for the composition of the HTTP request.

```java
package UD04.Ejemplo02;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class IMDb {

    public static StringBuilder htmlDownload(String address) throws Exception {
        StringBuilder answer = new StringBuilder();
        URL url = new URL(address);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Content-Type", "text/plain");

        connection.setRequestProperty("charset", "utf-8");
        connection.setRequestProperty("User-Agent", "Mozilla/5.0");
        int state = connection.getResponseCode();
        Reader streamReader = null;
        if (state == HttpURLConnection.HTTP_OK) {
            streamReader = new InputStreamReader(connection.getInputStream());
            int character;
            while ((character = streamReader.read()) != -1) {
                answer.append((char) character);
            }
        } else {
            throw new Exception("HTTP Error: " + state);
        }
        connection.disconnect();
        return answer;
    }

    public static void writeFile(String strPath, String content) throws IOException {
        Path path = Paths.get(strPath);
        byte[] strToBytes = content.getBytes();
        Files.write(path, strToBytes);
    }

    public static void main(String[] args) {
        try {
            String scheme = "https://";
            String server = "www.imdb.com";
            String path = "/find";
            String text = URLEncoder.encode("Tiburón",
                    StandardCharsets.UTF_8.name());
```

```
            String parameters = "?q=";
            String address = scheme + server + path + parameters + text;
            StringBuilder result = htmlDownload(address);
            IMDb.writeFile("src/UD04/Ejemplo02/tiburon.html", result.toString());
            System.out.println("Download completed");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

8.3. Example3

HTTP request with `java.net.http` (Java 11 and higher)

This example performs the same actions as Example1 but with classes from the `java.net.http` library.

```
package UD04.Ejemplo03;

import java.io.UnsupportedEncodingException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.HttpURLConnection;
import java.net.URI;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.nio.file.Path;

public class RAE2 {

    public static int htmlDownload(String address) throws Exception {
        URI url = new URI(address);
        HttpClient httpClient = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .followRedirects(HttpClient.Redirect.NORMAL)
                .build();
        HttpRequest request = HttpRequest.newBuilder()
                .GET()
                .uri(url)
                .headers("Content-Type", "text/plain")
                .setHeader("User-Agent", "Mozilla/5.0")
                .build();
        HttpResponse<Path> response =
httpClient.send(request,HttpResponse.BodyHandlers.ofFile(Path.of("src/UD04/Ejemplo03/ti
buron.html")));
        return response.statusCode();
    }

    public static void main(String[] args) {
```

```
        try {
            String scheme = "https://";
            String server = "dle.rae.es/";
            String resource = URLEncoder.encode("Tiburón",
                    StandardCharsets.UTF_8.name());
            String address = scheme + server + resource;
            int stateCode = htmlDownload(address);
            if (stateCode == HttpURLConnection.HTTP_OK) {
                System.out.println("Download completed");
            } else {
                System.err.println("Error: " + stateCode);
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

8.4. Example4

Upload and download of a file via FTP

This example uploads a file to the FTP server.

> This example requires the use of the `org.apache.commons` package that we have seen in theory.

```
package UD04.Ejemplo04;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.SocketException;
import org.apache.commons.net.ftp.*;

/**
 *
 * @author David Martínez (wwww.martinezpenya.es|www.eduardoprimo.es)
 */
public class FTPManager {

    private FTPClient FTPClient;
    private static final String SERVER = "ftp.dlptest.com";
    private static final int PORT = 21;
    private static final String USER = "dlpuser";
    private static final String PASSWORD = "rNrKYTX9g7z3RgJRmxWuGHbeu";

    public FTPManager() {
        FTPClient = new FTPClient();
```

```
    }

    private void connect() throws SocketException, IOException {
        FTPClient.connect(SERVER, PORT);
        int answer = FTPClient.getReplyCode();
        if (!FTPReply.isPositiveCompletion(answer)) {
            FTPClient.disconnect();
            throw new IOException("Error connecting to FTP Server");
        }
        boolean credentials = FTPClient.login(USER, PASSWORD);
        if (!credentials) {
            throw new IOException("Error connecting to FTP. Wrong credentials");
        }
        FTPClient.setFileType(FTP.BINARY_FILE_TYPE);
    }

    private void desconectar() throws IOException {
        FTPClient.disconnect();
    }

    private boolean uploadFile(String path) throws IOException {
        File localFile = new File(path);
        boolean sent;
        try ( InputStream is = new FileInputStream(localFile)) {
            sent = FTPClient.storeFile(localFile.getName(), is);
        }
        return sent;
    }

    public static void main(String[] args) {
        FTPManager ftpManager = new FTPManager();
        try {
            ftpManager.connect();
            System.out.println("Connected");
            boolean uploaded = ftpManager.uploadFile(
                    "src/UD04/Ejemplo04/upload.txt");
            if (uploaded) {
                System.out.println("File successfully uploaded.");
            } else {
                System.err.println("Something went wrong uploading file.");
            }
            ftpManager.desconectar();
            System.out.println("Disconnected");

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

8.5. Example5

Sending emails without attachments from a GMail account

This example is intended to send a text-only email using a Gmail account.

During execution, the program will ask the user for the recipient's and sender's email addresses and the password. This password is not encrypted on the screen, so you have to make sure that no one is watching.

The text of the message is encoded in the code.

> **Configure the institute's or personal gmail account to be able to use external applications (not secure)**
>
> Enter your account settings. Click on the icon with your start and choose the [Manage your Google account] button, access the [Security] section (on the left side of the screen). Then look for the option "Access less secure applications" and make sure it is enabled.

Once the mail is sent, it should appear in the mailbox of the recipient's account.

> This example requires the package `javax.mail` (JavaMail) and `javax.activation-1.2.0` as discussed in theory.

```
1  package UD04.Ejemplo05;
2
3  import java.io.IOException;
4  import java.util.Properties;
5  import java.util.Scanner;
6  import javax.mail.Message;
7  import javax.mail.MessagingException;
8  import javax.mail.NoSuchProviderException;
9  import javax.mail.Session;
10 import javax.mail.Transport;
11 import javax.mail.internet.AddressException;
12 import javax.mail.internet.InternetAddress;
13 import javax.mail.internet.MimeMessage;
14
15 public class EmailManager {
16
17     private Properties properties;
18     private Session session;
19
20     private void setSMTPServerProperties() {
21         properties = System.getProperties();
22         properties.put("mail.smtp.auth", "true");
23         properties.put("mail.smtp.host", "smtp.gmail.com");
24         properties.put("mail.smtp.port", 587);
25         properties.put("mail.smtp.starttls.enable", "true");
26         session = Session.getInstance(properties, null);
27     }
28
29     private Transport connectSMTPServer(String emailAddress, String password) throws
   NoSuchProviderException, MessagingException {
30         Transport t = (Transport) session.getTransport("smtp");
31         t.connect(properties.getProperty("mail.smtp.host"), emailAddress,
```

```
                password);
        return t;
    }

    private Message createMessageCore(String from, String to, String subject) throws
AddressException, MessagingException {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
        message.setSubject(subject);
        return message;
    }

    private Message createTextMessage(String from, String to, String subject,
            String messageText) throws MessagingException, AddressException,
IOException {
        Message message = createMessageCore(from, to, subject);
        message.setText(messageText);
        return message;
    }

    public void sendTextMessage(String from, String to, String subject,
            String messageText, String user, String password) throws AddressException,
MessagingException, IOException {
        setSMTPServerProperties();
        Message message = createTextMessage(from, to, subject, messageText);
        Transport t = connectSMTPServer(user, password);
        t.sendMessage(message, message.getAllRecipients());
        t.close();
    }

    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            System.out.print("Introduce email to:");
            String emailTo = sc.nextLine();
            System.out.print("Introduce email from:");
            String emailFrom = sc.nextLine();
            System.out.print("Introduce password:");
            String passwordFrom = sc.nextLine();
            sc.close();
            EmailManager emailManager = new EmailManager();
            emailManager.sendTextMessage(emailFrom, emailTo,
                    "Sample text email without attachment",
                    "Test Message from Java.",
                    emailFrom, passwordFrom);
            System.out.println("Email sent.");
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

```
81 }
```

8.6. Example6

Reading emails from the INBOX folder of a GMail email account

This example reads the inbox of a Gmail email account. For each available message, the sender's email address, the recipient's email address, the subject and the text of the message will be displayed if the content is only text. In the event that the email has attached files, a notice will be displayed instead of the text.

Activate IMAP in the institute's or personal gmail account to be able to read emails

The advantage of IMAP over POP3 is that reading the messages does not imply their removal from the server, so they will continue to be accessible later from any device. Because Gmail accounts are designed to be accessed from the web page, they are not configured to be used from external applications, access through IMAP and POP3 being disabled.

Access the GMail settings (icon of a gear wheel at the top right of the page) and click on the "See all settings" button. Within the "Forwarding and POP/IMAP mail" make sure that the "Enable IMAP" box is checked. is selected.

This example requires the package `javax.mail` (JavaMail) and `javax.activation-1.2.0` as discussed in theory.

```
1  package UD04.Ejemplo06;
2
3  import com.sun.mail.imap.IMAPFolder;
4  import java.util.Properties;
5  import java.util.Scanner;
6  import javax.mail.*;
7  import javax.mail.Message.*;
8  import javax.mail.internet.*;
9
10 public class EmailReader {
11
12     private Session getSesionImap() {
13         Properties properties = new Properties();
14         properties.setProperty("mail.store.protocol", "imap");
15         properties.setProperty("mail.imap.host", "imap.gmail.com");
16         properties.setProperty("mail.imap.port", "993");
17         properties.setProperty("mail.imap.ssl.enable", "true");
18         properties.setProperty("mail.imap.ssl.trust", "imap.gmail.com");
19         Session session = Session.getDefaultInstance(properties);
20         return session;
21     }
22
23     public void readINBOX(String email, String password) throws Exception {
24         Session session = this.getSesionImap();
25         Store storage = session.getStore("imaps");
26         storage.connect("imap.gmail.com", 993, email, password);
27         IMAPFolder inbox = (IMAPFolder) storage.getFolder("INBOX");
```

```
28          inbox.open(Folder.READ_WRITE);
29          Message[] messages = inbox.getMessages();
30          for (Message message : messages) {
31              Address[] fromAddress = message.getFrom();
32              String from = fromAddress[0].toString();
33              Address[] toAddress = message.getRecipients(RecipientType.TO);
34              String to = toAddress[0].toString();
35              String subject = message.getSubject();
36              MimeMultipart mimeMultipart = (MimeMultipart) message.getContent();
37              if (mimeMultipart.getBodyPart(0).isMimeType("text/plain")) {
38                  String textoMensaje = (String)
    mimeMultipart.getBodyPart(0).getContent();
39                  System.out.printf("From %s To %s Subject: %s Message: %s\n", from,
40                          to, subject, textoMensaje);
41              } else {
42                  System.out.printf("From %s To %s Subject: %s Message: %s\n", from,
43                          to, subject, "[*ATTACHED FILES*]");
44              }
45          }
46      }
47
48      public static void main(String[] args) {
49          Scanner sc = new Scanner(System.in);
50          System.out.print("Introduce your gmail address:");
51          String email = sc.nextLine();
52          System.out.print("Introduce your password:");
53          String password = sc.nextLine();
54          sc.close();
55          try {
56              new EmailReader().readINBOX(email, password);
57          } catch (Exception e) {
58              e.printStackTrace();
59          }
60      }
61  }
```

8.7. Example7

Here is an example that illustrates the process of creating and running a distributed application using Java RMI.

The system consists of two applications, client and server. Both share the same remote interface, in this case `Calculator`.

On the server side, there is the `Calculator` class, where the implementations of the remote methods are located, and the `Register` class, responsible for creating the registry (through the `createRegistry` method of the `LocateRegistry` class, indicating host and port) and to instantiate and register (the `bind` method of `Registry`) the remote objects, in this case the `Calculator` class.

On the client side, in addition to the interface, there is the `Client` class. In this class, the reference to the remote object registry is obtained (you must know the machine and the port in which it is located) to, from this, obtain the reference to the remote object through the lookup method. Once the reference has been obtained, the methods of the remote interface are called as if it were a local object.

Interface:

```
package UD04.Ejemplo07.interfaces;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CalculatorInterface extends Remote {
    public int add(int o1, int o2) throws RemoteException;
    public int substract(int o1, int o2) throws RemoteException;
    public int multiply(int o1, int o2) throws RemoteException;
    public int divide(int o1, int o2) throws RemoteException;
}
```

Register:

```
package UD04.Ejemplo07.server;

import UD04.Ejemplo07.interfaces.CalculatorInterface;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Register {

    public static void registerCalculator() {
        try {
            Registry register = LocateRegistry.createRegistry(5555);
            Server calculator = new Server();
            register.bind("Server",
                    (CalculatorInterface) UnicastRemoteObject.exportObject(
                            calculator, 0));
            System.out.println("Server ready...");
        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (AlreadyBoundException ex) {
            ex.printStackTrace();
        }

    }

    public static void main(String[] args) {
        registerCalculator();
```

```
    }
}
```

Server:

```
package UD04.Ejemplo07.server;

import UD04.Ejemplo07.interfaces.CalculatorInterface;
import java.rmi.RemoteException;

public class Server implements CalculatorInterface {

    @Override
    public int add(int o1, int o2) throws RemoteException {
        return o1 + o2;
    }

    @Override
    public int substract(int o1, int o2) throws RemoteException {
        return o1 - o2;
    }

    @Override
    public int multiply(int o1, int o2) throws RemoteException {
        return o1 * o2;
    }

    @Override
    public int divide(int o1, int o2) throws RemoteException {
        return o1 / o2;
    }
}
```

Client:

```
package UD04.Ejemplo07.client;

import UD04.Ejemplo07.interfaces.CalculatorInterface;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

    private CalculatorInterface calculator = null;

    public Client() {
        try {
            Registry registro = LocateRegistry.getRegistry("localhost", 5555);
            calculator = (CalculatorInterface) registro.lookup("Server");
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Client client = new Client();
        int result;
        try {
            result = client.calculator.add(34, 5);
            System.out.println(result);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
}
```

The execution consists of three phases:

- Starting the registry server by running the `rmiregistry` program.

```
$ rmiregistry
```

- Execution of the server, which will create and register the remote objects.

```
Server ready...
```

- Execution of the clients that obtain the references to the remote objects and use them by invoking their methods.

```
39
```

9. Information sources

- Wikipedia
- Programación de servicios y procesos - FERNANDO PANIAGUA MARTÍN [Paraninfo]
- Programación de Servicios y Procesos - ALBERTO SÁNCHEZ CAMPOS [Ra-ma]
- Programación de Servicios y Procesos - Mª JESÚS RAMOS MARTÍN - [Garceta] (1ª y 2ª Edición)
- Programación de servicios y procesos - CARLOS ALBERTO CORTIJO BON [Sintesis]
- Programació de serveis i processos - JOAR ARNEDO MORENO, JOSEP CAÑELLAS BORNAS i JOSÉ ANTONIO LEO MEGÍAS [IOC]
- GitHub repositories:
    - https://github.com/ajcpro/psp
    - https://oscarmaestre.github.io/servicios/index.html
    - https://github.com/juanro49/DAM/tree/master/DAM2/PSP
    - https://github.com/pablohs1986/dam_psp2021
    - https://github.com/Perju/DAM
    - https://github.com/eldiegoch/DAM
    - https://github.com/eldiegoch/2dam-psp-public
    - https://github.com/franlu/DAM-PSP
    - https://github.com/ProgProcesosYServicios
    - https://github.com/joseluisgs
    - https://github.com/oscarnovillo/dam2_2122
    - https://github.com/PacoPortillo/DAM_PSP_Tarea02_La-Cena-de-los-Filosofos