

Instituto Politecnico Nacional

Escuela Superior de Cómputo



Bocanegra Heziquio Yestlanezi

Practica 6

Sockets Servidores

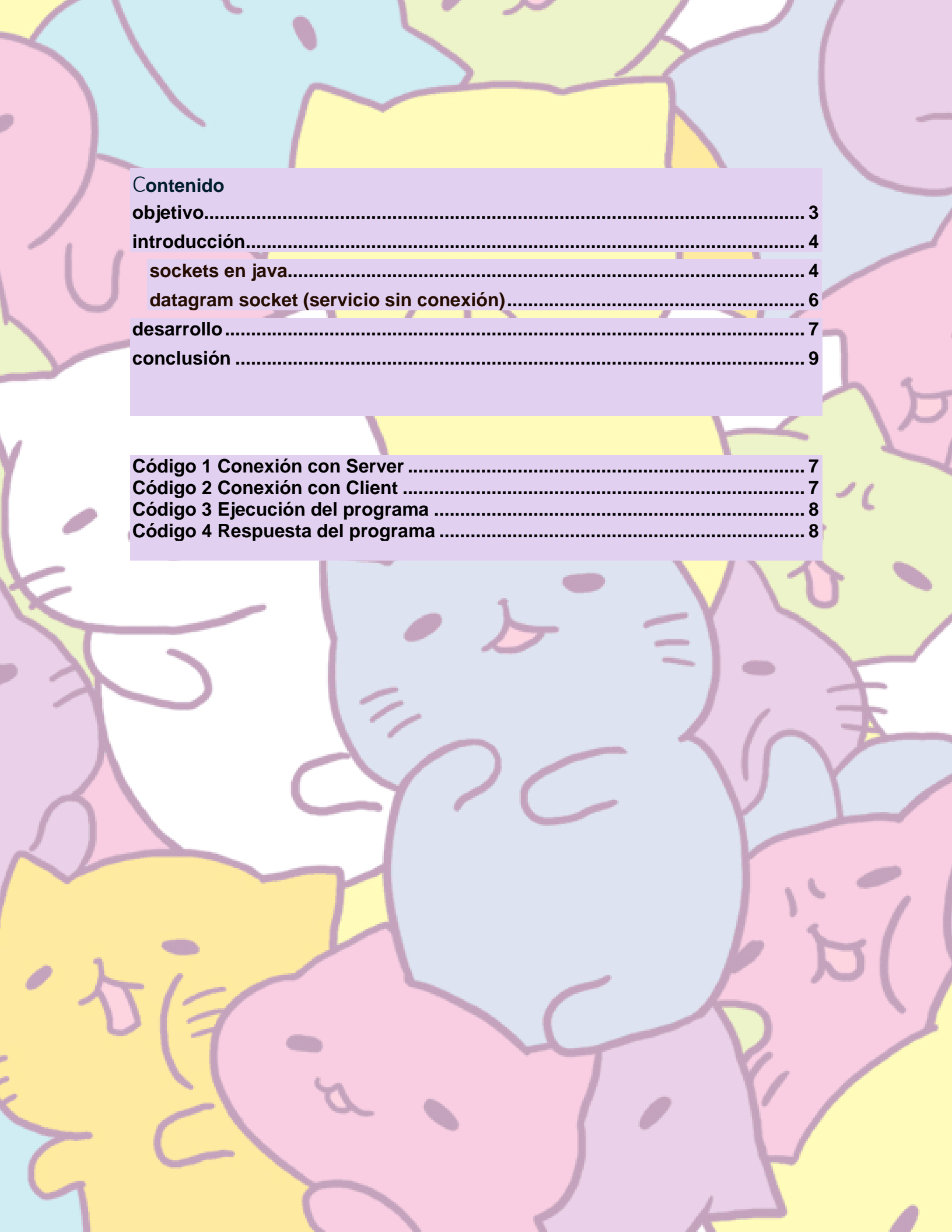
I. ChatBot Básico

Fecha: 11 de junio de 2021

2CM13

Programación Orientada a Objetos





Contenido	
objetivo.....	3
introducción.....	4
sockets en java.....	4
datagram socket (servicio sin conexión).....	6
desarrollo	7
conclusión	9

Código 1 Conexión con Server	7
Código 2 Conexión con Client	7
Código 3 Ejecución del programa	8
Código 4 Respuesta del programa	8

The background of the entire page is a dense, colorful collage of cartoon cat faces. The cats are drawn in a simple, cute style with large eyes and various expressions. The colors used for the cats include shades of pink, purple, blue, yellow, and green. Some cats are looking forward, while others are looking to the side or have their mouths open as if meowing. The overall aesthetic is playful and whimsical.

OBJETIVO

Codificar un cliente y un servidor que interactúe del siguiente modo el cliente envía una respuesta al servidor y el servidor envía una respuesta al cliente. El servidor puede almacenar al menos 10 preguntas y 10 respuestas predefinidas (se pueden usar 2 arreglos o un HashMap). Si el cliente le pide al servidor que cuente un chiste entre 10 chistes disponibles elige uno al azar y lo envía, si el cliente le pide al servidor la hora entonces que el servidor envíe la hora actual.

INTRODUCCIÓN

SOCKETS EN JAVA

La programación en red siempre ha sido dificultosa, el programador debía de conocer la mayoría de los detalles de la red, incluyendo el hardware utilizado, los distintos niveles en que se divide la capa de red, las librerías necesarias para programar en cada capa, etc.

Pero, la idea simplemente consiste en obtener información desde otra maquina, aportada por otra aplicación software.

Por lo tanto, de cierto modo se puede reducir al mero hecho de leer y escribir archivos, con ciertas salvedades.

El sistema de Entrada/Salida de Unix sigue el paradigma que normalmente se designa como Abrir-Leer-Escribir-Cerrar.

Antes de que un proceso de usuario pueda realizar operaciones de entrada/salida, debe hacer una llamada a Abrir (open) para indicar, y obtener los permisos del fichero o dispositivo que se desea utilizar.

Una vez que el fichero o dispositivo se encuentra abierto, el proceso de usuario realiza una o varias llamadas a Leer (read) y Escribir (write), para la lectura y escritura de los datos

El proceso de lectura toma los datos desde el objeto y los transfiere al proceso de usuario, mientras que el de escritura los transfiere desde el proceso de usuario al objeto. Una vez concluido el intercambio de información, el proceso de usuario llamará a Cerrar (close) para informar al sistema operativo que ha finalizado la utilización del fichero o dispositivo.

En Unix, un proceso tiene un conjunto de descriptores de entrada/salida desde donde leer y por donde escribir. Estos descriptores pueden estar referidos a ficheros, dispositivos, o canales de comunicaciones sockets.

El ciclo de vida de un descriptor, aplicado a un canal de comunicación (por ejemplo, un socket), está determinado por tres fases :

Creación, apertura del socket

Lectura y Escritura, recepción y envío de datos por el socket

Destrucción, cierre del socket

La interface IPC en Unix-BSD está implementada sobre los protocolos de red TCP y UDP. Los destinatarios de los mensajes se especifican como direcciones de socket; cada dirección de socket es un identificador de comunicación que consiste en una dirección Internet y un número de puerto

Las operaciones IPC se basan en pares de sockets.

Se intercambian información transmitiendo datos a través de mensajes que circulan entre un socket en un proceso y otro socket en otro proceso. Cuando los mensajes

son enviados, se encolan en el socket hasta que el protocolo de red los haya transmitido.

Cuando llegan, los mensajes son encolados en el socket de recepción hasta que el proceso que tiene que recibirlos haga las llamadas necesarias para recoger esos datos.

El lenguaje Java fue desarrollado por la empresa Sun Microsystems hacia el año 1990, mediante la creación de un grupo de trabajo en cuya cabeza estaba James Gosling. Este grupo de trabajo fue ideado para desarrollar un sistema de control de electrodomésticos y de PDAs o asistentes personales (pequeños ordenadores) y que además tuviese la posibilidad de interconexión a redes de ordenadores. Todo ello implicaba la creación de un hardware polivalente, un sistema operativo eficiente (SunOS) y un lenguaje de desarrollo (Oak). El proyecto concluyó dos años más tarde con un completo fracaso que condujo a la disolución del grupo.

Cuando se escriben programas Java que se comunican a través de la red, se está programando en la capa de aplicación. Típicamente, no se necesita trabajar con las capas TCP y UDP, en su lugar se puede utilizar las clases del paquete `java.net`. Estas clases proporcionan comunicación de red independiente del sistema.

A través de las clases del paquete `java.net`, los programas Java pueden utilizar TCP o UDP para comunicarse a través de Internet. Las clases `URL`, `URLConnection`, `Socket`, y `SocketServer` utilizan TCP para comunicarse a través de la Red. Las clases `DatagramPacket` y `DatagramServer` utilizan UDP.

TCP proporciona un canal de comunicación fiable punto a punto, lo que utilizan para comunicarse las aplicaciones cliente-servidor en Internet. Las clases `Socket` y `ServerSocket` del paquete `java.net` proporcionan un canal de comunicación independiente del sistema utilizando TCP, cada una de las cuales implementa el lado del cliente y el servidor respectivamente.

Así el paquete `java.net` proporciona, entre otras, las siguientes clases, que son las que se verán con detalle:

Socket: Implementa un extremo de la conexión TCP.

- ♥ **ServerSocket:** Se encarga de implementar el extremo Servidor de la conexión en la que se esperarán las conexiones de los clientes.
- ♥ **DatagramSocket:** Implementa tanto el servidor como el cliente cuando se utiliza UDP.

- ♥ DatagramPacket: Implementa un datagram packet, que se utiliza para la creación de servicios de reparto de paquetes sin conexión.
- ♥ InetAddress: Se encarga de implementar la dirección IP.

La clase Socket del paquete java.net es una implementación independiente de la plataforma de un cliente para un enlace de comunicación de dos vías entre un cliente y un servidor. Utilizando la clase java.net.Socket en lugar de tratar con código nativo, los programas Java pueden comunicarse a través de la red de una forma independiente de la plataforma.

El entorno de desarrollo de Java incluye un paquete, java.io, que contiene un juego de canales de entrada y salida que los programas pueden utilizar para leer y escribir datos. Las clases InputStream y OutputStream del paquete java.io son superclases abstractas que definen el comportamiento de los canales de I/O de tipo stream de Java. java.io también incluye muchas subclases de InputStream y OutputStream que implementan tipos específicos de canales de I/O.

DATAGRAM SOCKET (Servicio sin Conexión)

Es el más simple, lo único que se hace es enviar los datos, mediante la creación de un socket y utilizando los métodos de envío y recepción apropiados. Se trata de un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la fiabilidad: los datos se envían y reciben en paquetes, cuya entrega no está garantizada; los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

DESARROLLO

```

C:\Users\yestl\Videos - java ChatServer
C:\Users\yestl\Videos\Practica 6>javac -cp .;j3dcore.jar;j3dutils.jar;vecmath.jar Tamagochi.java
C:\Users\yestl\Videos\Practica 6>java -cp .;j3dcore.jar;j3dutils.jar;vecmath.jar Tamagochi
Esperando por el servidor . . .
Conectado al servidor.
run

C:\Users\yestl\Videos\Practica 6>cd ..
C:\Users\yestl\Videos>cd practica 6
C:\Users\yestl\Videos\Practica 6>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: 96DE-20EA

Directorio de C:\Users\yestl\Videos\Practica 6

11/06/2021 12:52 a. m.      <DIR>          .
11/06/2021 12:52 a. m.      <DIR>          ..
23/02/2021 08:27 p. m.      2,280 ChatClient.class
23/02/2021 08:27 p. m.      1,695 ChatClient.java
23/02/2021 08:27 p. m.      3,762 ChatServer.class
23/02/2021 08:27 p. m.      3,481 ChatServer.java
23/02/2021 08:27 p. m.      4 archivos      11,138 bytes
                        2 dirs 31,525,302,272 bytes libres

C:\Users\yestl\Videos\Practica 6>javac ChatServer.java
C:\Users\yestl\Videos\Practica 6>java ChatServer
^C
C:\Users\yestl\Videos\Practica 6>javac ChatServer.java
C:\Users\yestl\Videos\Practica 6>java ChatServer
^C
C:\Users\yestl\Videos\Practica 6>javac ChatServer.java
C:\Users\yestl\Videos\Practica 6>java ChatServer
^C
C:\Users\yestl\Videos\Practica 6>javac ChatServer.java
C:\Users\yestl\Videos\Practica 6>java ChatServer

```

Código 1 Conexión con Server

[illegible]

Código 2 Conexión con Client

CONCLUSIÓN

Con esta práctica queríamos crear un código que contara con un servidor y un cliente que pudieran interactuar, de modo que el cliente debía ser capaz de enviar una respuesta al servidor y el servidor debía ser capaz de enviar una respuesta al cliente.

En el desarrollo de la práctica, en las capturas de pantalla del símbolo de sistema (CMD) se puede observar que existe la conexión entre el Server y el Client, que estarán interactuando, se cuenta con preguntas predefinidas y a estas se les debe dar la respuesta que igualmente ya estaba predefinida, así como el usuario puede elegir la opción de contar un chiste y el programa arroja un chiste aleatorio de los que ya están predefinidos en el código.

Con base en lo aprendido puedo concluir que la práctica fue realizada con satisfacción, ya que el programa funcionó y relativamente hace lo solicitado.