TP : JAVA NIO

Exercice 1 : Copie de fichiers

L'objectif de cet exercice est d'utiliser les classes FileChannel et ByteBuffer pour copier le contenu d'un fichier source (sourceCh.txt) vers un fichier de destination (destinationCh.txt).

```
import java.io.*;
import java.nio.*;
import java.nio.channels.FileChannel;

public class FileChannelExample {
    public static void main(String[] args) {
       try (FileChannel sourceChannel = new
     FileInputStream("sourceCh.txt").getChannel();
                  FileChannel destChannel = new
     FileOutputStream("destinationCh.txt").getChannel()) {

            ByteBuffer buffer = ByteBuffer.allocate(1024);

            while (sourceChannel.read(buffer) != -1) {
                buffer.flip();
                destChannel.write(buffer;
                buffer.clear();
            }

            System.out.println("File copied successfully!");

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Exercice 2 : Sockets et JAVA NIO.

Supposant que nous avons le code côté serveur et le code côté client ci-dessous.

Travail à faire : faire l'implémentation des 2 fonctions *sendDataToServer et readDataFromServer (Côté client).*

```
public class Server {
    public static void main(String[] args) {
        try {
            // Open a selector
            Selector selector = Selector.open();

            // Open a server channel and configure it to be non-blocking
            ServerSocketChannel serverSocketChannel =
ServerSocketChannel.open();
            serverSocketChannel.bind(new InetSocketAddress(4444));
            serverSocketChannel.configureBlocking(false);

            // Register the server channel with the selector for accepting
connections
            serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);

            System.out.println("Server started on port 4444...");

            // Main server loop
            while (true) {
                // Select ready channels
                selector.select();

                // Get the set of selected keys
                Set<SelectionKey> selectedKeys = selector.selectedKeys();
                Iterator<SelectionKey> keyIterator =
selectedKeys.iterator();

                while (keyIterator.hasNext()) {
                    SelectionKey key = keyIterator.next();

                    if (key.isAcceptable()) {
                        // Accept an incoming connection
    ServerSocketChannel serverChannel = (ServerSocketChannel) key.channel();
    SocketChannel clientChannel = serverChannel.accept();
    clientChannel.configureBlocking(false);

                        // Register the client channel for read operations
          clientChannel.register(selector, SelectionKey.OP_READ);
```

```
                        System.out.println("Client connected: " +
clientChannel.getRemoteAddress());
                    } else if (key.isReadable()) {
                        // Handle reading data
                        SocketChannel clientChannel = (SocketChannel)
key.channel();
                        ByteBuffer buffer = ByteBuffer.allocate(1024);
                        int bytesRead = clientChannel.read(buffer);

                        if (bytesRead > 0) {
                            buffer.flip();
                            byte[] data = new byte[buffer.remaining()];
                            buffer.get(data);
                            System.out.println("Received from "
                                    + clientChannel.getRemoteAddress() + ":
" + new String(data));

                            // Respond to the client
                            String response = "Hello, Client!";

clientChannel.write(ByteBuffer.wrap(response.getBytes()));
                        } else if (bytesRead == -1) {
                            // Connection has been closed by the client
                            key.cancel();
                            clientChannel.close();
                            System.out.println("Client disconnected: " +
clientChannel.getRemoteAddress());
                        }
                    }

                    // Remove the key to avoid processing it again
                    keyIterator.remove();
                }
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
                                          }
```

```java
public class Client {
    public static void main(String[] args) {
        try (SocketChannel socketChannel = SocketChannel.open()) {
            // Configure the client socket to be non-blocking
            socketChannel.configureBlocking(false);

            // Connect to the server
            socketChannel.connect(new InetSocketAddress("localhost", 4444));

            // Wait for the connection to be established
            while (!socketChannel.finishConnect()) {

            }

            System.out.println("Connected to the server. Type 'exit' to 
quit.");

            // Start reading user input and sending it to the server
            try (Scanner scanner = new Scanner(System.in)) {
                while (true) {
                    System.out.print("Enter message: ");
                    String userInput = scanner.nextLine();

                    if ("exit".equalsIgnoreCase(userInput)) {
                        break;
                    }

                    sendDataToServer(socketChannel, userInput);

                    readDataFromServer(socketChannel);
                }
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```