Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

Algoritmos y Estructuras de Datos I (CE 1103)

I Semestre 2024

Tarea Extra-Clase 2

“CLEAN CODE”

Integrantes del Equipo:

Dylan Elizondo Picado 2023086320

Harold Madriz Cerdas 2023106162

Darien Golfín Tencio 2023152232

Fecha de Entrega:24/4/2024

1. Nomenclatura

Código Original:

```
using System.Net.Sockets;

namespace Sockets{
    public class ClientSocket{
        private int port;
        private TcpClient client;
        private NetworkStream stream;
        ...
    }
}
```

```
public class JSONGenerator
{
    private static readonly ILogger<JSONGenerator> _logger = Logger.CreateLogger<JSONG
    ...
}
```

Problema:

- Las variables **port**, **client** y **stream** podrían tener nombres más descriptivos para clarificar su propósito sin necesidad de comentar o revisar el tipo de datos.

- La variable _logger puede tener un nombre más específico relacionado a su uso en la clase.

Corrección:

```
namespace Sockets{
    public class ClientSocket{
        private int serverPort;
        private TcpClient tcpClientConnection;
        private NetworkStream tcpStream;
        ...
    }
}
```

```
public class JSONGenerator
{
    private static readonly ILogger<JSONGenerator> jsonLogger = Logger.CreateLogger<JSO
    ...
}
```

2. Funciones

Código Original:

```
public string ProcessData(string data){
    try {
        byte[] buf = Encoding.UTF8.GetBytes(data + "\n");
        ...
    }
    catch (IOException ex) {
        _logger.LogError("Error de E/S: " + ex.Message);
        return "";
    }
}
```

```
public void Down(string song, ClientSocket clientSock){
    JSONGenerator json = new JSONGenerator();
    Dictionary<string, object> data = new Dictionary<string, object>{
        {"command", "down.vote"},
        {"id", song}
    };
    string voto = json.GenerateJSON(data);
    clientSock.ProcessData(voto);
    _logger.LogInformation($"Voting down for song: {song}");
}
```

Problema:

- La función **ProcessData** está realizando varias tareas: convierte datos, envía datos, recibe respuesta y maneja errores.
- La función **Down** está realizando tanto la generación de JSON como la comunicación de red y el logging, mezclando lógicas que deberían estar separadas.

Corrección:

```
public void SendData(string data){
    byte[] buffer = Encoding.UTF8.GetBytes(data + "\n");
    stream.Write(buffer, 0, buffer.Length);
}

public string ReceiveData(){
    byte[] buffer = new byte[1600];
    int bytesRead = stream.Read(buffer, 0, 1600);
    return Encoding.UTF8.GetString(buffer, 0, bytesRead).Trim();
}

public string ProcessData(string data){
    try {
        SendData(data);
        return ReceiveData();
    }
    catch (IOException ex) {
        _logger.LogError("Error de E/S: " + ex.Message);
        return "";
    }
}
```

```
public void VoteOnSong(string songId, string command, ClientSocket clientSock)
    string payload = PrepareVotePayload(songId, command);
    clientSock.ProcessData(payload);
}

private string PrepareVotePayload(string songId, string command){
    JSONGenerator json = new JSONGenerator();
    Dictionary<string, object> data = new Dictionary<string, object>{
        {"command", command},
        {"id", songId}
    };
    return json.GenerateJSON(data);
}
```

3. Comentarios

Código Original:

```
public static async Task Request(ClientSocket clientSock, CancellationToken cancellationToken,
    var config = new IniFile("config.ini"); //Variable de ruta archivo configuracion
    int delay = Convert.ToInt32(config.Read("Delay"));
    ...
}
```

```
// Método para obtener un nodo específico en la lista
public Node getNode(int index) {
    ...
}
```

Problema:

- Comentario innecesario que describe lo obvio.
- El comentario es redundante porque el nombre del método getNode ya es bastante descriptivo sobre lo que hace el método.

Corrección: Eliminar el comentario y asegurarse que el nombre de la variable sea descriptivo:

```
public static async Task Request(ClientSocket clientSock, CancellationToken
    var configFilePath = "config.ini";
    var config = new IniFile(configFilePath);
    int delayInSeconds = Convert.ToInt32(config.Read("Delay"));
    ...
}
```

```
public Node getNode(int index) {
    ...
}
```

4. Formato

Código Original:

```
public void startServer() {
    try (ServerSocket serverSocket = new ServerSocket(port, backlog)) {
        System.out.println("Started Listening for clients");
        while (true) {
            // take input and output streams
            try (Socket client = serverSocket.accept();
                 Scanner scanner = new Scanner(client.getInputStream());
                 PrintWriter pw = new PrintWriter(client.getOutputStream(), true))
                String dataFromClient = scanner.nextLine();
                String response = getResponse(dataFromClient);
                pw.write(response);
            } catch(Exception e) {
                e.printStackTrace();
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public void processData(String data)
{
    byte[] buf;
    /* append newline as server expects a line to be read */
    buf = Encoding.UTF8.GetBytes(data+"\n");
    Console.WriteLine("Sending data \'{0}\' to server", data);
    stream.Write(buf, 0, data.Length + 1);
    buf = new byte[100];
    int bytesRead = stream.Read(buf, 0, 100);
    byte[] finalData = new byte[bytesRead];
    for(int i=0; i < bytesRead; i++)
    {
        finalData[i] = buf[i];
    }
    string response = Encoding.UTF8.GetString(finalData);
    response = response.TrimEnd();
    Console.WriteLine("Received Response : \'{0}\', of length {1}", response, respon
    client.Close();
}
```

Problema:

- El manejo de excepciones y los flujos de control están demasiado anidados, lo que puede dificultar la lectura y el mantenimiento del código.

- El código tiene varios problemas de formato: las líneas están muy juntas, hay un uso inconsistente de espacios y la estructura de bucles no es clara.

Corrección:

```
public void startServer() {
    try (ServerSocket serverSocket = new ServerSocket(port, backlog)) {
        System.out.println("Started Listening for clients");
        handleConnections(serverSocket);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private void handleConnections(ServerSocket serverSocket) throws IOException {
    while (true) {
        try (Socket client = serverSocket.accept();
             Scanner scanner = new Scanner(client.getInputStream());
             PrintWriter pw = new PrintWriter(client.getOutputStream(), true)) {
            String dataFromClient = scanner.nextLine();
            String response = getResponse(dataFromClient);
            pw.write(response);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
public void processData(string data)
{
    byte[] buffer = Encoding.UTF8.GetBytes(data + "\n");
    Console.WriteLine($"Sending data '{data}' to server");

    stream.Write(buffer, 0, buffer.Length);
    buffer = new byte[100];
    int bytesRead = stream.Read(buffer, 0, 100);

    string response = Encoding.UTF8.GetString(buffer, 0, bytesRead).TrimEnd();
    Console.WriteLine($"Received Response: '{response}', of length {response.Length}"

    client.Close();
}
```

5. Objetos y estructuras de datos

Código Original:

```
public class ListDoubleO {
    private Node tail; // Puntero al último nodo en la lista
    private int size = 0; // Llevara cuenta del tamaño
    ...
}
```

```
public class ClientSocket
{
    private int port;
    private TcpClient client;
    private NetworkStream stream;

    public ClientSocket(int port, int sendTimeout, int receiveTimeout)
    {
        this.port = port;
        client = new TcpClient("localhost", port);
        client.ReceiveTimeout = sendTimeout;
        client.SendTimeout = receiveTimeout;
        stream = client.GetStream();
    }

    public void processData(String data)
    {
        byte[] buf = Encoding.UTF8.GetBytes(data + "\n");
        Console.WriteLine("Sending data \'{0}\' to server", data);
        stream.Write(buf, 0, data.Length + 1);
        buf = new byte[100];
        int bytesRead = stream.Read(buf, 0, 100);
        string response = Encoding.UTF8.GetString(buf, 0, bytesRead).TrimEnd();
        Console.WriteLine("Received Response : \'{0}\', of length {1}", response, re
        client.Close();
    }
}
```

Problema:

- La clase expone detalles internos como **tail** y **size** que deberían estar encapsulados para prevenir manipulaciones no controladas.
- La clase ClientSocket gestiona la conexión y también procesa datos, lo que infringe el principio de responsabilidad única.

Corrección:

```
public class ListDoubleO {
    private Node tail;
    private int size = 0;

    public int getSize() {
        return size;
    }

    public Song getSongAt(int index) {
        Node node = getNode(index);
        return node != null ? node.data : null;
    }
    ...
}
```

```
public class ClientConnection
{
    private TcpClient client;
    private NetworkStream stream;

    public ClientConnection(string hostname, int port)
    {
        client = new TcpClient(hostname, port);
    }

    public NetworkStream Stream
    {
        get { return client.GetStream(); }
    }

    public void Close()
    {
        client.Close();
    }
}

public class ClientSocket
{
    private ClientConnection connection;

    public ClientSocket(string hostname, int port)
    {
        connection = new ClientConnection(hostname, port);
    }

    public void ProcessData(string data)
    {
        var buffer = Encoding.UTF8.GetBytes(data + "\n");
        Console.WriteLine($"Sending data '{data}' to server");
        connection.Stream.Write(buffer, 0, buffer.Length);

        buffer = new byte[100];
        int bytesRead = connection.Stream.Read(buffer, 0, 100);
        string response = Encoding.UTF8.GetString(buffer, 0, bytesRead).TrimEnd();

        Console.WriteLine($"Received Response: '{response}', of length {response.Le
        connection.Close();
    }
}
```

6. Manejo de errores

Código Original:

```
try {
    ClientSocket clientSock = new ClientSocket(1234, 3000, 3000);
    String data = "Hello World";
    clientSock.processData(data);
} catch(Exception e) {
    Console.WriteLine(e.Message);
}
```

```
public void startServer() {
    try (ServerSocket serverSocket = new ServerSocket(port, backlog)) {
        System.out.println("Started Listening for clients");
        while (true) {
            try (Socket client = serverSocket.accept();
                 Scanner scanner = new Scanner(client.getInputStream());
                 PrintWriter pw = new PrintWriter(client.getOutputStream(), true)) {
                 String dataFromClient = scanner.nextLine();
                 String response = getResponse(dataFromClient);
                 pw.write(response);
             }catch(Exception e){
                 e.printStackTrace();
             }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Problema:

- El manejo de excepciones es genérico y no proporciona una forma clara de responder a diferentes tipos de errores.
- No escribir correctamente las sentencias try-catch-finally.

Corrección:

```
try {
    ClientSocket clientSock = new ClientSocket(1234, 3000, 3000);
    String data = "Hello World";
    clientSock.processData(data);
} catch (SocketException e) {
    Console.WriteLine("Error de conexión: " + e.Message);
} catch (IOException e) {
    Console.WriteLine("Error de E/S: " + e.Message);
} catch (Exception e) {
    Console.WriteLine("Error no esperado: " + e.Message);
}
```

```
public void startServer() {
    try (ServerSocket serverSocket = new ServerSocket(port, backlog)) {
        System.out.println("Started Listening for clients");
        while (true) {
            handleClientConnection(serverSocket);
        }
    } catch (IOException e) {
        System.err.println("Server Socket error: " + e.getMessage());
    }
}

private void handleClientConnection(ServerSocket serverSocket) {
    try (Socket client = serverSocket.accept();
         Scanner scanner = new Scanner(client.getInputStream());
         PrintWriter pw = new PrintWriter(client.getOutputStream(), true)) {
        String dataFromClient = scanner.nextLine();
        String response = getResponse(dataFromClient);
        pw.write(response);
    } catch (SocketTimeoutException e) {
        System.err.println("Socket timed out: " + e.getMessage());
    } catch (IOException e) {
        System.err.println("I/O error: " + e.getMessage());
    }
}
```