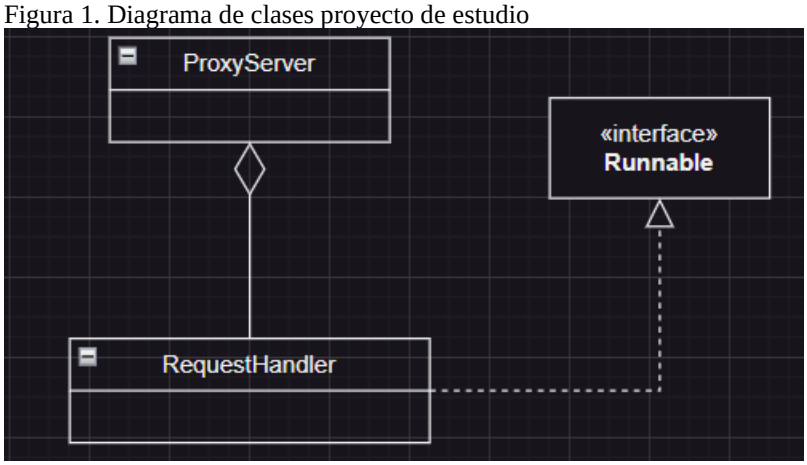
Taller #5
Patrones

Alison Aristizabal – 202125179

1. URL proyecto de estudio: https://github.com/wundulo/proxy/tree/master

   Servidor proxy multiproceso Java que puede manejar solicitudes GET y POST simples. Este proxy también elimina la información del encabezado de conexión del proxy y del agente de usuario para navegar en modo incógnito.

   Figura 1. Diagrama de clases proyecto de estudio

   - Retos de diseño
     - El código tiene varios IOException sin manejo.
     - El servidor proxy no tiene ninguna validación de seguridad lo que es riesgoso para el mismo.
     - El manejo de concurrencia es deficiente.
     - Siempre asume que la operación de Socket es exitosa, por lo cual no maneja una excepción en un caso diferente.

2. .

   a. Clase ProxyServer

```
public ProxyServer (int port) {

        executor = Executors.newCachedThreadPool();
        try { server = new ServerSocket(port); }
        catch (IOException e) {     }
}

/**
 * Create new socket and request handler object on each request
 *
 */
public void accept() {

        while (true) {
                try { executor.execute(new RequestHandler(server.accept())); }
                catch (IOException e) {     }
```

```
        }
    }

    /**
     * Main method to fire up proxy server and launch request handlers
     *
     */
    public static void main(String[] args) {

        System.out.println("ProxyServer is listening to port "+LISTEN_PORT);
        ProxyServer proxy = new ProxyServer(LISTEN_PORT);
        proxy.accept();
    }
```

b. Clase RequestHandler

```
public void run() {

        try {

            clientInputStream = new DataInputStream(clientSocket.getInputStream());
            clientOutputStream = clientSocket.getOutputStream();

            // step 1) get request from client
            clientToProxy();

            // step 2) forward request to remote host
            proxyToRemote();

            // step 3) read response from remote back to client
            remoteToClient();

            System.out.println();

            if(remoteOutputStream != null) remoteOutputStream.close();
            if(remoteInputStream != null) remoteInputStream.close();
            if(remoteSocket != null) remoteSocket.close();


            if(clientOutputStream != null) clientOutputStream.close();
            if(clientInputStream != null) clientInputStream.close();
            if(clientSocket != null) clientSocket.close();

        } catch (IOException e) { }
    }
```

ProxyServer actúa como un proxy que toma un puerto y crea una instancia de RequestHandler para manejar las solicitudes entrantes.
RequestHandler procesa la solicitud del cliente y se comunica con el servidor remoto.

3. El patrón que se utiliza es el patrón Proxy. El patrón se centra en la mediación entre un objeto y otro, este permite realizar acciones antes y después de realizar la acción deseada por el usuario. El usuario interactúa con el Proxy.

4. La clase ProxyServer actúa como el proxy que controla el acceso y crea la instancia de RequestHandler que es la clase que realiza el trabajo y se comunica con el servidor remoto.

5. Tiene sentido utilizar el patrón Proxy en este proyecto ya que proporciona mayor segmentación y flexibilidad para el manejo de solicitudes y la interacción con el servidor. Tiene como ventaja la eficiencia, el control de acceso, encapsulamiento y escalabilidad.

6. El proyecto tiene una dependencia directa de las operaciones “DataInputStream”, “OutputStream” que podría hacer más difícil introducir cambios en la implementación sin afectar el resto del código.
   checkClientStreams() y checkRemoteStreams() manejan una reconnexion simplificada que no tiene en cuenta problemas de desconexión o de red.

7. Para la concurrencia se podría haber creado manualmente un sistema de gestión de concurrencia.
   Dentro de RequestHandler se pudo haber aplicado directamente la implementación del filtrado de cabeceras y la simplificación del manejo de solicitudes y sus respuestas.