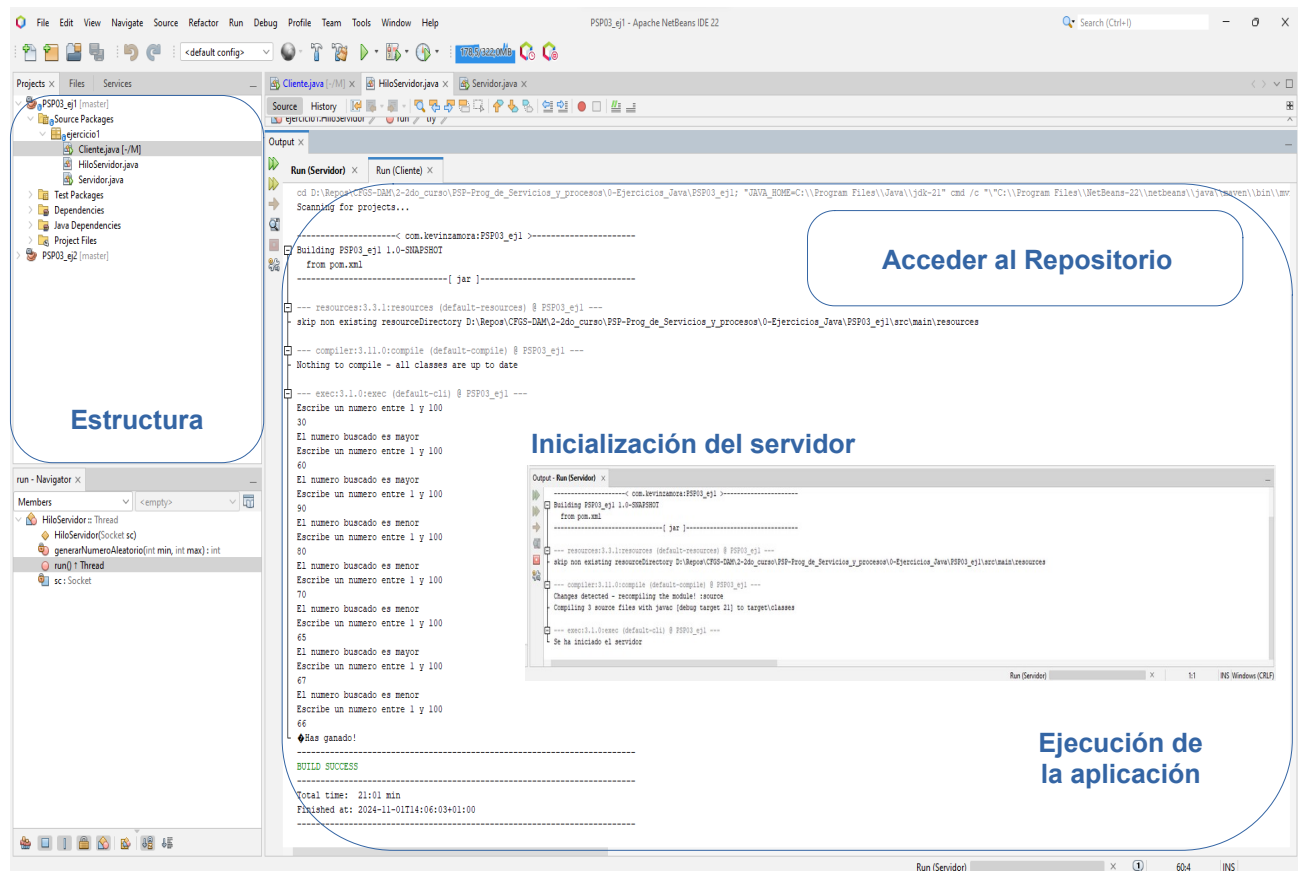


Ejercicio 1

- Creamos la estructura del proyecto e inicializamos las diferentes clases/entidades necesarias (“Cliente”, “HiloServidor” y “Servidor”).



Seguidamente, en las páginas siguientes, se muestra en detalle el desarrollo de los/as tres componentes/clases/entidades desarrolladas. Y junto con su desarrollo, también se pueden apreciar los comentarios y las “etiquetas”, para generar la documentación con la herramienta Java Doc, que se han añadido.

Desarrollamos la entidad '**Ciente**' y añadimos los comentarios con Java Doc.

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package ejercicio1;
6
7  import java.io.DataInputStream;
8  import java.io.DataOutputStream;
9  import java.io.IOException;
10 import java.net.ServerSocket;
11 import java.net.Socket;
12 import java.util.Scanner;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15
16 /**
17  *
18  * Clase/Entidad Ciente.
19  * @author kzdesigner
20  */
21 public class Ciente {
22
23     /**
24      * Método 'main'/principal encargado de iniciar el lado del cliente
25      * @param args Argumentos
26      */
27     public static void main(String[] args) {
28
29         /** Método de control de excepciones 'try catch' */
30         try {
31
32             /** Definimos un 'socket'/zócalo para nuestro servidor y lo ubicamos
33              * en el puerto 2000 */
34             Socket sc = new Socket("localhost", 2000);
35
36             /** Definimos e inicializamos los flujos de datos de entrada y salida */
37             DataInputStream in = new DataInputStream(sc.getInputStream());
38             DataOutputStream out = new DataOutputStream(sc.getOutputStream());
39
40             /** Definimos la variable booleana para la función 'salir' */
41             boolean salir = false;
42
43             /** Definimos e inicializamos el medio de entrada de texto por teclado */
44             Scanner scanner = new Scanner(System.in);
45
46             /** Bucle 'haz mientras' */
47             do {
48
49                 /** Leemos el número/mensaje construido desde el HiloServidor */
50                 String mensaje = in.readUTF();
51
52                 /** Imprimimos por pantalla el mensaje recibido */
53                 System.out.println(mensaje);
54
55                 /** Leemos el número entero solicitado por pantalla */
56                 int num = scanner.nextInt();
57                 /** Escribimos/Enviamos dicho número al 'hilo'/interlocutor */
58                 out.writeInt(num);
59
60                 /** Cargamos el mensaje generado desde la clase HiloServidor */
61                 mensaje = in.readUTF();
62                 /** E imprimimos dicho mensaje. Sus valores serán: 'verdadero'
63                  * cuando ambos números sean iguales o 'falso' cuando sean diferentes */
64                 System.out.println(mensaje);
65
66                 /** Leemos el valor del booleano recibido para saber cuándo
67                  * se ha adivinado el número 'secreto' generado y salir. */
68                 salir = in.readBoolean();
69
70             } while (!salir); /* Mientras 'salir' sea falso, se repetirá el bucle */
71
72             sc.close(); /* Cerramos la conexión */
73
74         } catch (IOException ex) { // Capturamos los posibles errores en un 'logger'
75             Logger.getLogger(Ciente.class.getName()).log(Level.SEVERE, null, ex);
76         }
77     }
78 }
79
80 }
```

Desarrollamos la entidad '**Servidor**' y añadimos los comentarios con Java Doc.

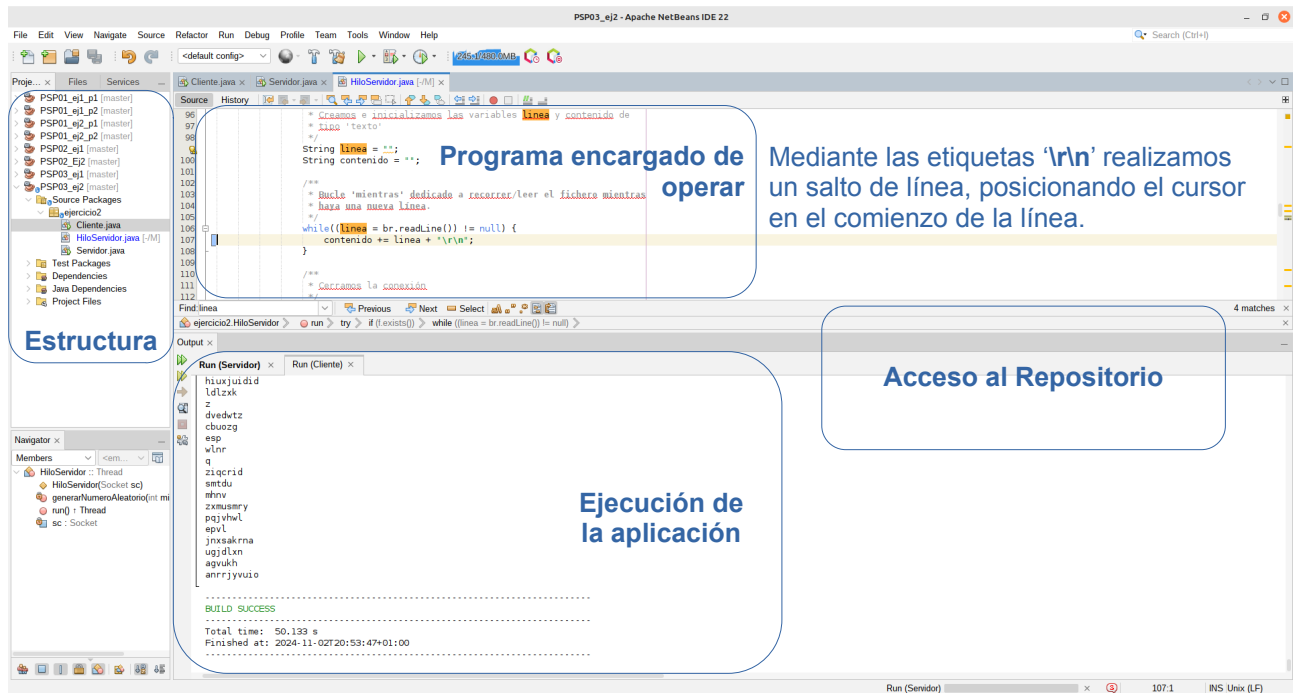
```
1 /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5 package ejercicio1;
6
7 import ejercicio1.HiloServidor;
8 import ejercicio1.Cliente;
9 import java.io.IOException;
10 import java.net.ServerSocket;
11 import java.net.Socket;
12 import java.util.logging.Level;
13 import java.util.logging.Logger;
14
15 /**
16  *
17  * Clase / Entidad Servidor.
18  * @author kzdesigner
19  */
20 public class Servidor {
21
22     /**
23      * Método 'main' dedicado a construir y habilitar el servidor.
24      * @param args Argumentos
25      */
26     public static void main(String[] args) {
27
28         /**
29          * Método de control de excepciones 'try catch'.
30          */
31         try {
32             // Creación de un 'zócalo' para alojar el servidor en el puerto 2000
33             ServerSocket servidor = new ServerSocket(2000);
34             System.out.println("Se ha iniciado el servidor");
35
36             // Bucle 'mientras sea verdadero'
37             while(true) {
38
39                 // Llamada al método 'acceptar()' del objeto servidor y guardado
40                 // del 'resultado' en una variable 'sc' de tipo 'Socket'
41                 Socket sc = servidor.accept();
42
43                 // Instancia/creación de un objeto de la entidad 'HiloServidor',
44                 // encargada de ejecutar y definir el funcionamiento y las etapas
45                 // de nuestra aplicación y a la cual le pasamos la aceptación del
46                 // servidor llamada 'sc'
47                 HiloServidor hs = new HiloServidor(sc);
48                 // Ejecución del método 'start' del objeto 'hs'
49                 hs.start();
50
51             }
52
53             } catch (IOException ex) { // Control y captura de los posibles errores
54                 Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
55             }
56
57     }
58
59 }
60
```

Desarrollamos la entidad 'HiloServidor' y añadimos los comentarios con Java Doc.

```
1 /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this licens
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5 package ejercicio1;
6
7 import java.io.DataInputStream;
8 import java.io.DataOutputStream;
9 import java.io.IOException;
10 import java.net.Socket;
11 import java.util.logging.Level;
12 import java.util.logging.Logger;
13
14 /**
15  *
16  * Clase/entidad HiloServidor que extiende a la clase nativa 'Hilo'
17  * @author kzdesigner
18  */
19 public class HiloServidor extends Thread {
20
21     /**
22      * Definimos el zócalo 'socket' de conexión
23      */
24     private Socket sc;
25
26     /**
27      * Creamos el constructor de clase a utilizar y definimos que le podamos
28      * pasar el 'zócalo' como parámetro.
29      * @param sc Zócalo de conexión (o Socket)
30      */
31     public HiloServidor(Socket sc) {
32         this.sc = sc;
33     }
34
35     /**
36      * Método 'run()' encargado de conectar el cliente y el servidor y también
37      * de realizar todas , las operaciones de entrada, salida y comparación
38      * necesarias.
39      */
40     public void run() {
41
42         /*
43          Imprimimos un mensaje informativo y creamos e inicializamos los
44          flujos de entrada y de salida.
45          */
46         System.out.println("Cliente conectado");
47         // Creación e inicialización de variables
48         DataInputStream in = null;
49         DataOutputStream out = null;
50         // Definimos un método de control de excepciones 'try catch' para
51         // capturar los posibles errores durante la ejecución del programa
52         try {
53             // Leemos y asignamos los datos recibidos desde los 'flujos de entrada
54             // y salida y los guardamos en las variables 'in' y 'out'
55             in = new DataInputStream(sc.getInputStream());
56             out = new DataOutputStream(sc.getOutputStream());
57
58             // Generamos el número aleatorio y lo guardamos en una variable
59             int numAleatorio = generarNumeroAleatorio(1,100);
60             // Inicializamos la variable 'numUsuario' a cero
61             int numUsuario = 0;
62
63             // Mostramos el número aleatorio generado
64             System.out.println("Num generado: " + numAleatorio);
65
66             // Bucle 'Haz mientras'. Este solicita un número por teclado
67             // hasta que se adivina el número 'secreto' generado en el 'servidor'
68             do {
69
70                 out.writeUTF("Escribe un numero entre 1 y 100");
71
72                 // Leemos el entero vía teclado
73                 numUsuario = in.readInt();
74
75                 // Mostramos el número introducido
76                 System.out.println("Numero recibido: " + numUsuario);
77
78                 // Bucle 'si / si no'. En este se compara el valor introducido
79                 // con el generado por el método 'calcularNumeroAleatorio'
80                 // alojado en la clase 'HiloServidor' y se muestra un mensaje
81                 if(numUsuario == numAleatorio){
82                     out.writeUTF("¡Has ganado!");
83                 }else if (numUsuario < numAleatorio){
84                     out.writeUTF("El numero buscado es mayor");
85                 }else{
86                     out.writeUTF("El numero buscado es menor");
87                 }
88                 // Evaluamos si el número del usuario y el generado son iguales
89                 out.writeBoolean(numUsuario == numAleatorio);
90
91                 // Mientras el número introducido y el generado sean diferentes
92                 // se ejecutará reiteradamente el programa anterior
93             } while(numUsuario != numAleatorio);
94
95             // Cerramos la conexión e imprimimos el mensaje correspondiente
96             sc.close();
97             System.out.println("Cliente desconectado");
98
99         } catch (IOException ex) { // Capturamos los errores y los guardamos en un 'logger'
100             Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
101         } finally {
102             try {
103                 // Cerramos los flujos de entrada y de salida
104                 in.close();
105                 out.close();
106             } catch (IOException ex) { // Capturamos los posibles errores en un logger
107                 Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
108             }
109         }
110     }
111 }
112
113 /**
114  * Método generarNumeroAleatorio dedicado a la creación de números al azar
115  * @param min Valor mínimo
116  * @param max Valor máximo
117  * @return num Número aleatorio generado
118  */
119 private int generarNumeroAleatorio(int min, int max) {
120     int num = (int) (Math.random()*(max - min + 1) + (min));
121     return num;
122 }
123
124 }
```

Ejercicio 2

- Creamos la estructura del proyecto e inicializamos las diferentes clases/entidades necesarias (“Cliente”, “Servidor” e “HiloServidor”).



Seguidamente, en las páginas siguientes, se muestra en detalle el desarrollo de los/as tres componentes/clases/entidades desarrolladas. Y junto con su desarrollo, también se pueden apreciar los comentarios y las “etiquetas”, para generar la documentación con la herramienta Java Doc, que se han añadido.

Desarrollamos la entidad 'Cliente' y añadimos los comentarios con Java Doc.

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package ejercicio2;
6
7  import java.io.DataInputStream;
8  import java.io.DataOutputStream;
9  import java.io.IOException;
10 import java.net.ServerSocket;
11 import java.net.Socket;
12 import java.util.Scanner;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15
16 /**
17  *
18  * Clase/Entidad Cliente
19  * @author kzdesigner
20  */
21 public class Cliente {
22
23     /**
24      * Método 'main' dedicado a definir y ejecutar toda la información necesaria
25      * para el funcionamiento del lado del cliente en un proceso de comunicación.
26      * @param args Argumentos
27      */
28     public static void main(String[] args) {
29
30         /** Método 'try catch' dedicado al control de excepciones */
31         try {
32
33             /** Creamos una instancia/llamada hacia la clase 'Socket'(o zócalo) y
34              * definimos el puerto y la dirección de conexión mediante uno de sus
35              * constructores, introduciendo en este dichos dos parámetros
36              */
37             Socket sc = new Socket("localhost", 1500);
38
39             /** Creamos e inicializamos las variables dedicadas a crear los
40              * flujos de datos de entrada y salida */
41             DataInputStream in = new DataInputStream(sc.getInputStream());
42             DataOutputStream out = new DataOutputStream(sc.getOutputStream());
43
44             /** Creamos y definimos la variable de la función 'salir' en 'falso' */
45             boolean salir = false;
46
47             /** Creamos y inicializamos un objeto del tipo Scanner dedicado a la
48              * recogida de datos a través del teclado */
49             Scanner scanner = new Scanner(System.in);
50             /** Usamos el carácter '\n' para definir un salto de línea */
51             scanner.useDelimiter("\n");
52
53             System.out.println("Introduce la ruta del archivo a mostrar:");
54             /** Leemos la ruta del archivo a leer introducida mediante teclado */
55             String ruta = scanner.next();
56
57             /** Escribimos la ruta introducida enviándola hacia el servidor */
58             out.writeUTF(ruta);
59
60             /** Leemos la variable 'existe' para definir si se debe un archivo */
61             boolean existe = in.readBoolean();
62
63             /** Evaluamos si existe el archivo */
64             if (existe) {
65
66                 /** Inicializamos la variable 'longitud' y leemos el valor entero
67                  * por teclado */
68                 int longitud = in.readInt();
69
70                 /** Creamos una colección de tipo 'byte' y creamos una variable
71                  * de este tipo, introduciendo y guardando la longitud del
72                  * archivo en su interior */
73                 byte[] contenido = new byte[longitud];
74
75                 /** Bucle para recorrer el archivo y leer cada línea, leyendo el
76                  * 'byte' que la representa a nivel informático */
77                 for (int i = 0; i < longitud; i++) {
78                     contenido[i] = in.readByte();
79                 }
80
81                 /** Creación de la variable 'contenidoFichero' y volcado del
82                  * contenido del fichero guardándolo en formato 'texto' */
83                 String contenidoFichero = new String(contenido);
84
85                 /** Imprimimos el contenido del fichero */
86                 System.out.println(contenidoFichero);
87
88             } else {
89                 /** Definimos un mensaje alternativo para cuando no haya archivo */
90                 System.out.println("Error, no existe el fichero");
91             }
92
93             /** Cerramos el zócalo de conexión */
94             sc.close();
95
96         } catch (IOException ex) {
97             /** Capturamos los posibles errores y los registramos en un 'Logger' */
98             Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
99         }
100     }
101 }
102
103 }
```

Desarrollamos la entidad '**Servidor**' y añadimos los comentarios con Java Doc.

```
1 /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5 package ejercicio2;
6
7 import ejercicio2.HiloServidor;
8 import ejercicio2.Cliente;
9 import java.io.IOException;
10 import java.net.ServerSocket;
11 import java.net.Socket;
12 import java.util.logging.Level;
13 import java.util.logging.Logger;
14
15 /**
16  *
17  * Clase/Entidad Servidor dedicada a iniciar nuestro servidor.
18  * @author kzdesigner
19  */
20 public class Servidor {
21
22     /**
23      * Método 'main' dedicado a establecer e iniciar nuestro servidor.
24      * @param args Argumentos
25      */
26     public static void main(String[] args) {
27
28         /**
29          * Definimos un método de control de excepciones 'try catch'
30          */
31         try {
32             /**
33              * Definimos una variable 'servidor' de tipo 'ServerSocket' y le pasamos
34              * como parámetro el puerto 1500.
35              */
36             ServerSocket servidor = new ServerSocket(1500);
37             System.out.println("Se ha iniciado el servidor");
38
39             /**
40              * Bucle 'mientras' para ejecutar el programa mientras sea verdadero.
41              */
42             while(true) {
43
44                 /** Definición de la variable sc, a la cual se le asigna el
45                  * método 'aceptar' del servidor. */
46                 Socket sc = servidor.accept();
47
48                 /** Creamos una instancia de la clase HiloServidor. */
49                 HiloServidor hs = new HiloServidor(sc);
50                 /** Implementamos el método 'start' del objeto 'hs' */
51                 hs.start();
52
53             }
54
55             } catch (IOException ex) { /** Capturamos las excepciones y las
56              * guardamos en un 'Logger' */
57                 Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
58             }
59
60     }
61
62 }
63
```

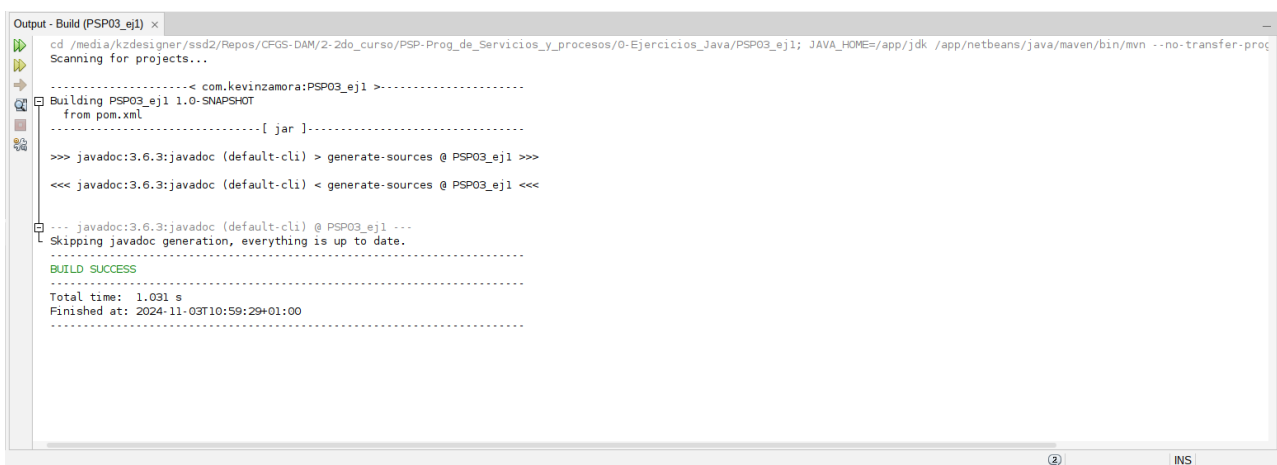

Desarrollamos la entidad 'HiloServidor' y añadimos los comentarios con Java Doc.

```
1 /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5 package ejercicio2;
6
7 import java.io.BufferedReader;
8 import java.io.DataInputStream;
9 import java.io.DataOutputStream;
10 import java.io.File;
11 import java.io.FileReader;
12 import java.io.IOException;
13 import java.net.Socket;
14 import java.util.logging.Level;
15 import java.util.logging.Logger;
16
17 /**
18  *
19  * Clase/Entidad HiloServidor que extiende a la clase nativa o tipo 'Hilo'
20  * @author kzdesigner
21  */
22 public class HiloServidor extends Thread {
23
24     /**
25      * Definición de la variable 'sc' de tipo 'socket'/(o zócalo), para la conexión
26      */
27     private Socket sc;
28
29     public HiloServidor(Socket sc) {
30         this.sc = sc;
31     }
32
33     /**
34      * Método sobrescrito 'run()', encargado de ejecutar el funcionamiento
35      * principal de nuestra aplicación y también de establecer la conexión entre
36      * cliente y servidor.
37      *
38      */
39     @Override
40     public void run() {
41
42         /**
43          * Mostramos el mensaje 'cliente conectado' e inicializamos los flujos
44          * de entrada y de salida de datos mediante las variables 'in' y 'out'.
45          */
46         System.out.println("Cliente conectado");
47         DataInputStream in = null;
48         DataOutputStream out = null;
49         /**
50          * Método de control de excepciones 'try catch'
51          */
52         try {
53             /**
54              * Asignación e inicialización de los flujos de entrada/salida a las
55              * variables.
56              */
57             in = new DataInputStream(sc.getInputStream());
58             out = new DataOutputStream(sc.getOutputStream());
59
60             /**
61              * Creación de una variable 'ruta' de tipo 'texto' y guardado de la
62              * ruta del archivo a leer en su interior, la cual ha sido solicitada
63              * previamente por teclado.
64              */
65             String ruta = in.readUTF();
66
67             /**
68              * Creación de una variable de tipo/clase nativo/a 'Fichero/File' y
69              * creación de un nuevo objeto 'File' pasándole el parámetro 'ruta'.
70              * @param ruta
71              */
72             File f = new File(ruta);
73
74             /**
75              * Sentencia condicional 'if else/si sino' para comprobar si existe
76              * o no un archivo que leer ubicado en la ruta introducida.
77              */
78             if (f.exists()) {
79                 /**
80                  * Si existe el archivo, escribimos un booleano con valor
81                  * verdadero en el servidor, para indicar que sí se ha detectado.
82                  */
83                 out.writeBoolean(true);
84
85                 /**
86                  * Definimos una variable 'br' de la clase nativa 'BufferedReader'
87                  * y creamos una instancia de dicho archivo pasándole como
88                  * parámetro la creación de un nuevo 'FileReader', la cual le
89                  * pasa a su vez la variable 'ruta' como parámetro. Dicha
90                  * variable 'br' es la encargada de establecer la conexión con
91                  * el fichero a leer y poder proceder así con su lectura.
92                  */
93                 BufferedReader br = new BufferedReader(new FileReader(ruta));
94
95                 /**
96                  * Creamos e inicializamos las variables linea y contenido de
97                  * tipo 'texto'
98                  */
99                 String linea = "";
100                 String contenido = "";
101
102                 /**
103                  * Bucle 'mientras' dedicado a recorrer/leer el fichero mientras
104                  * haya una nueva línea.
105                  */
106                 while ((linea = br.readLine()) != null) {
107                     contenido += linea;
108                 }
109
110                 /**
111                  * Cerramos la conexión
112                  */
113                 br.close();
114
115                 /**
116                  * Creamos una colección de tipo 'byte' y guardamos en esta el
117                  * contenido del archivo convertida a 'bytes'.
118                  */
119                 byte[] contenidoFichero = contenido.getBytes();
120
121                 /**
122                  * Escribimos la longitud del archivo y se la enviamos al servidor.
123                  * @param contenidoFichero.length Longitud del fichero
124                  */
125                 out.writeInt(contenidoFichero.length);
126
127                 /**
128                  * Bucle 'para' dedicado a recorrer/leer el archivo línea por
129                  * línea o mejor dicho: 'byte' por 'byte'.
130                  */
131                 for (int i = 0; i < contenidoFichero.length; i++) {
132                     out.writeByte(contenidoFichero[i]);
133                 }
134
135                 /**
136                  * Cerramos la conexión
137                  */
138                 sc.close();
139
140             } else {
141                 /**
142                  * En caso de no encontrar el archivo, escribimos/enviamos un
143                  * valor 'falso'.
144                  */
145                 out.writeBoolean(false);
146
147                 /**
148                  * Cerramos la conexión e imprimimos el mensaje correspondiente.
149                  */
150                 sc.close();
151                 System.out.println("Cliente desconectado");
152
153             }
154         } catch (IOException ex) { /** Controlamos las excepciones y guardamos
155                                  la información de estos en un 'Logger'. */
156             Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
157         } finally {
158             /**
159              * Definimos un método de control de excepciones 'try catch'.
160              */
161             try {
162                 /**
163                  * Cerramos los flujos de datos de entrada y de salida.
164                  */
165                 in.close();
166                 out.close();
167             } catch (IOException ex) {
168                 /**
169                  * Guardamos la información de los errores en un 'Logger'.
170                  */
171                 Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
172             }
173         }
174     }
175 }
176
177 /**
178  * Método 'generarNumeroAleatorio' dedicado a la función que se indica con
179  * su nombre.
180  * @param min Valor mínimo
181  * @param max Valor máximo
182  * @return num Número generado
183  */
184 private int generarNumeroAleatorio(int min, int max) {
185     int num = (int) (Math.random() * (max - min + 1) + (min));
186     return num;
187 }
188
189 }
190
```


Con lo anteriormente expuesto, la presente tarea ya está prácticamente terminada, ahora sólo faltaría generar la documentación ejecutando 'Java Doc' y exportar los archivos comprimidos de ambos proyectos, con extensión .zip.

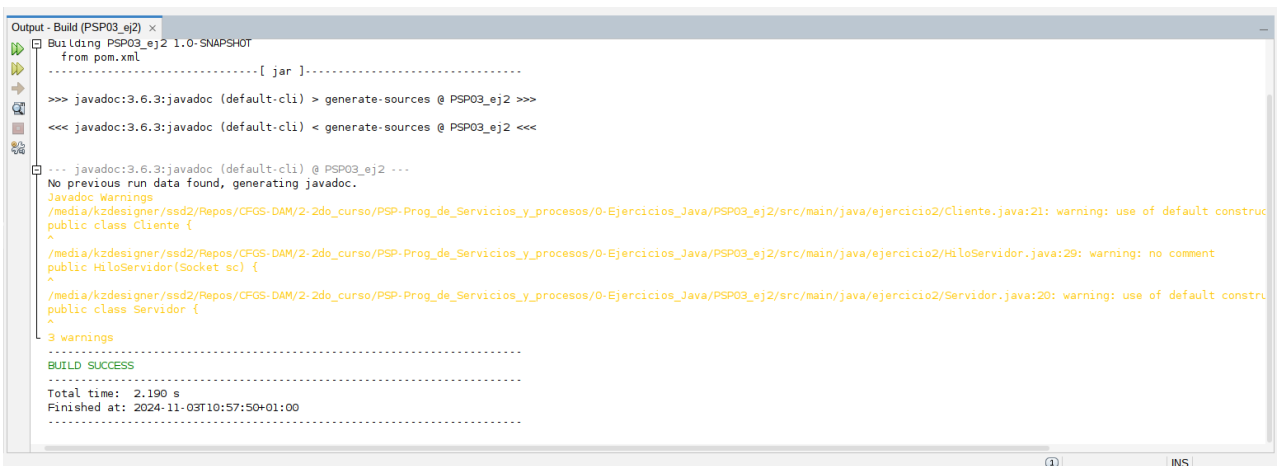
Para generar la documentación 'Java Doc' de cada uno de los proyectos, seleccionamos el directorio raíz de cada uno y hacemos clic derecho sobre este. Acto seguido, se nos desplegará un menú y entre sus primeras opciones, recorriéndolo de arriba a abajo, encontraremos la opción 'Generar Java Doc' (o *Generate Java Doc*). Al hacer clic sobre la citada opción, se empezará a ejecutar el correspondiente proceso y nos mostrará por pantalla resultados similares a los de las capturas de pantalla siguientes:

Generación de la documentación del ejercicio 1:



```
Output - Build (PSP03_ej1) x
cd /media/kzdesigner/ssd2/Repos/CFGs-DAM/2-2do_curso/PSP-Prog_de_Servicios_y_procesos/0-Ejercicios_Java/PSP03_ej1; JAVA_HOME=/app/jdk /app/netbeans/java/maven/bin/mvn --no-transfer-proc
Scanning for projects...
Building PSP03_ej1 1.0-SNAPSHOT
from pom.xml
[ jar ]
>>> javadoc:3.6.3:javadoc (default-cli) > generate-sources @ PSP03_ej1 >>>
<<< javadoc:3.6.3:javadoc (default-cli) < generate-sources @ PSP03_ej1 <<<
--- javadoc:3.6.3:javadoc (default-cli) @ PSP03_ej1 ---
Skipping javadoc generation, everything is up to date.
BUILD SUCCESS
Total time: 1.031 s
Finished at: 2024-11-03T10:59:29+01:00
```

Generación de la documentación del ejercicio 2:



```
Output - Build (PSP03_ej2) x
Building PSP03_ej2 1.0-SNAPSHOT
from pom.xml
[ jar ]
>>> javadoc:3.6.3:javadoc (default-cli) > generate-sources @ PSP03_ej2 >>>
<<< javadoc:3.6.3:javadoc (default-cli) < generate-sources @ PSP03_ej2 <<<
--- javadoc:3.6.3:javadoc (default-cli) @ PSP03_ej2 ---
No previous run data found, generating javadoc.
Javadoc Warnings
/media/kzdesigner/ssd2/Repos/CFGs-DAM/2-2do_curso/PSP-Prog_de_Servicios_y_procesos/0-Ejercicios_Java/PSP03_ej2/src/main/java/ejercicio2/Cliente.java:21: warning: use of default construc
public class Cliente {
^
/media/kzdesigner/ssd2/Repos/CFGs-DAM/2-2do_curso/PSP-Prog_de_Servicios_y_procesos/0-Ejercicios_Java/PSP03_ej2/src/main/java/ejercicio2/HiloServidor.java:29: warning: no comment
public HiloServidor(Socket sc) {
^
/media/kzdesigner/ssd2/Repos/CFGs-DAM/2-2do_curso/PSP-Prog_de_Servicios_y_procesos/0-Ejercicios_Java/PSP03_ej2/src/main/java/ejercicio2/Servidor.java:20: warning: use of default constru
public class Servidor {
^
3 warnings
BUILD SUCCESS
Total time: 2.190 s
Finished at: 2024-11-03T10:57:50+01:00
```

El siguiente y último paso consiste en acceder a la documentación generada. Para ello, accedemos al directorio '**target/site/apidocs/**', partiendo desde el directorio raíz de nuestro proyecto, y una vez estamos en este, procedemos a abrir el archivo '**index.html**' con nuestro navegador web.

Al abrir dicho archivo '**index.html**' con nuestro navegador web elegido (Mozilla Firefox, Google Chrome, Opera, Brave, etc.), aparecerá una aplicación web como la siguiente.

Como en este proyecto sólo contamos con un paquete y este contiene las tres entidades/clases desarrolladas, nuestra 'página principal' (en cuanto a la documentación de ambos ejercicios), presenta directamente las diferentes clases, desarrolladas en nuestra aplicación:

PACKAGE

CLASS

USE

TREE

INDEX

HELP

PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES

SEARCH

Package ejercicio1

package ejercicio1

Classes

Class	Description
Cliente	Clase/Entidad Cliente.
HiloServidor	Clase/Entidad HiloServidor que extiende a la clase nativa 'hilo'
Servidor	Clase / Entidad Servidor.

Desde esta 'vista', podemos inspeccionar las diferentes clases del paquete 'ejercicio1(o 2)' y, al acceder a la clase seleccionada, nos aparecerá una 'vista en detalle' de todas sus propiedades, métodos y otros aspectos de interés. En la siguiente página se muestra una captura de cada una de las clases en cuestión.

Página con los detalles principales de la clase **Cliente**:

PACKAGE

CLASS

USE

TREE

INDEX

HELP

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHODSEARCH

Package `ejercicio1`

Class Cliente

`java.lang.Object`
`ejercicio1.Cliente`

```
public class Cliente
extends Object
```

Clase/Entidad Cliente.

Author:
kzdesigner

Constructor Summary

Constructors

Constructor	Description
<code>Cliente()</code>	

Method Summary

All Methods

Static Methods

Concrete Methods

Modifier and Type	Method	Description
static void	<code>main(String[] args)</code>	Método 'main' dedicado a establecer la conexión con el servidor y recoger el mensaje con el número generado en el servidor.

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Details

Cliente

```
public Cliente()
```

Method Details

main

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

Método 'main' dedicado a establecer la conexión con el servidor y recoger el mensaje con el número generado en el servidor.

Parameters:
args - Argumentos de tipo 'cadena de texto'

Y a su vez, al acceder a cada una de las clases, también tendremos acceso a las secciones: **‘uso’** (si está usada o no), **‘árbol/tree’** (donde se indexa la estructura de directorios/carpetas), **‘index/índice’** (donde se muestra qué es cada objeto/método) y finalmente **‘ayuda’** (donde se muestra la sección de ayuda), donde se muestra información específica sobre cada una de las clases e información más genérica, en la sección ‘ayuda’.

Página ‘uso’:

PACKAGECLASSUSETREEINDEXHELP

SEARCH

Search

Uses of Class
ejercicio1.Cliente

Página ‘árbol’:

PACKAGECLASSUSETREEINDEXHELP

SEARCH

Search

Hierarchy For Package ejercicio1

Class Hierarchy

- java.lang.Object
 - ejercicio1.Cliente
 - ejercicio1.Servidor
 - java.lang.Thread (implements java.lang.Runnable)
 - ejercicio1.HiloServidor

Página ‘Índice’:

PACKAGECLASSUSETREETREEINDEXHELP

SEARCH

Search

Index

C E H M R S
All Classes and Interfaces | All Packages

C

Cliente - Class in ejercicio1
Clase/Entidad Cliente.
Cliente() - Constructor for class ejercicio1.Cliente

E

ejercicio1 - package ejercicio1

H

HiloServidor - Class in ejercicio1
Clase/Entidad HiloServidor que extiende a la clase nativa 'hilo'
HiloServidor(Socket) - Constructor for class ejercicio1.HiloServidor
Constructor de la clase HiloServidor

M

main(String[]) - Static method in class ejercicio1.Cliente
Método 'main' dedicado a establecer la conexión con el servidor y recoger el mensaje con el número gene
main(String[]) - Static method in class ejercicio1.Servidor
Método 'main' dedicado a construir y habilitar el servidor.

R

run() - Method in class ejercicio1.HiloServidor
Método 'run()' dedicado a ejecutar las diferentes 'etapas' de nuestra aplicación

S

Servidor - Class in ejercicio1
Clase / Entidad Servidor.
Servidor() - Constructor for class ejercicio1.Servidor

C E H M R S

Página ‘Ayuda’:

OVERVIEWPACKAGECLASSUSETREETREEINDEXHELP

HELP: NAVIGATION | PAGES

SEARCH

Search

JavaDoc Help

- Navigation:
 - Search
- Kinds of Pages:
 - Overview
 - Package
 - Class or Interface
 - Other Files
 - Use
 - Tree (Class Hierarchy)
 - All Packages
 - All Classes and Interfaces
 - Index

Navigation

Starting from the Overview page, you can browse the documentation using the links in each page, and in the navigation bar at the top of each page. The Index and Search box allow you to navigate to specific declarations and summary pages, including: All Packages, All Classes and Interfaces

Search

You can search for definitions of modules, packages, types, fields, methods, system properties and other terms defined in the API. These items can be searched using part or all of the name, optionally using "camelCase" abbreviations, or multiple search terms separated by whitespace. Some examples:

- "j.l.obj" matches "java.lang.Object"
- "InpStr" matches "java.io.InputStream"
- "math exact long" matches "java.lang.Math.absExact(long)"

Refer to the Javadoc Search Specification* for a full description of search features.

Kinds of Pages

The following sections describe the different kinds of pages in this collection.

Overview

The Overview page is the front page of this API document and provides a list of all packages with a summary for each. This page can also contain an overall description of the set of packages.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. These pages may contain the following categories:

- Interfaces
- Classes
- Enum Classes
- Exception Classes
- Annotation Interfaces

Página con los detalles principales de la clase **HiloServidor**:

PACKAGECLASSUSETREINDEXHELP

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHODSEARCHSearch

Package ejercicio1

Class HiloServidor

java.lang.Object[Ⓜ]
java.lang.Thread[Ⓜ]
ejercicio1.HiloServidor

All Implemented Interfaces:
Runnable[Ⓜ]

public class HiloServidor
extends Thread[Ⓜ]

Clase/Entidad HiloServidor que extiende a la clase nativa 'hilo'

Author:
kzdesigner

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread[Ⓜ]

Thread.Builder[Ⓜ], Thread.State[Ⓜ], Thread.UncaughtExceptionHandler[Ⓜ]

Field Summary

Fields inherited from class java.lang.Thread[Ⓜ]

MAX_PRIORITY[Ⓜ], MIN_PRIORITY[Ⓜ], NORM_PRIORITY[Ⓜ]

Constructor Summary

Constructors

Constructor	Description
HiloServidor(Socket [Ⓜ] sc)	Constructor de la clase HiloServidor

Method Summary

All Methods**Instance Methods****Concrete Methods**

Modifier and Type	Method	Description
void	run()	Método 'run()' dedicado a ejecutar las diferentes 'etapas' de nuestra aplicación

Methods inherited from class java.lang.Thread[Ⓜ]
activeCount[Ⓜ], checkAccess[Ⓜ], clone[Ⓜ], countStackFrames[Ⓜ], currentThread[Ⓜ], dumpStack[Ⓜ], enumerate[Ⓜ], getAllStackTraces[Ⓜ], getContextClassLoader[Ⓜ], getDefaultUncaughtExceptionHandler[Ⓜ], getId[Ⓜ], getName[Ⓜ], getPriority[Ⓜ], getStackTrace[Ⓜ], getState[Ⓜ], getThreadGroup[Ⓜ], getUncaughtExceptionHandler[Ⓜ], holdsLock[Ⓜ], interrupt[Ⓜ], interrupted[Ⓜ], isAlive[Ⓜ], isDaemon[Ⓜ], isInterrupted[Ⓜ], isVirtual[Ⓜ], join[Ⓜ], join[Ⓜ], join[Ⓜ], join[Ⓜ], ofPlatform[Ⓜ], ofVirtual[Ⓜ], onSpinWait[Ⓜ], resume[Ⓜ], setContextClassLoader[Ⓜ], setDaemon[Ⓜ], setDefaultUncaughtExceptionHandler[Ⓜ], setName[Ⓜ], setPriority[Ⓜ], setUncaughtExceptionHandler[Ⓜ], sleep[Ⓜ], sleep[Ⓜ], sleep[Ⓜ], start[Ⓜ], startVirtualThread[Ⓜ], stop[Ⓜ], suspend[Ⓜ], threadId[Ⓜ], toString[Ⓜ], yield[Ⓜ]

Methods inherited from class java.lang.Object[Ⓜ]
equals[Ⓜ], finalize[Ⓜ], getClass[Ⓜ], hashCode[Ⓜ], notify[Ⓜ], notifyAll[Ⓜ], wait[Ⓜ], wait[Ⓜ], wait[Ⓜ]

Constructor Details

HiloServidor

public HiloServidor(Socket[Ⓜ] sc)

Constructor de la clase HiloServidor

Parameters:
sc - Variable 'zócalo' pasado al constructor como parámetro

Method Details

run

public void run()

Método 'run()' dedicado a ejecutar las diferentes 'etapas' de nuestra aplicación

Specified by:
run[Ⓜ] in interface Runnable[Ⓜ]

Overrides:
run[Ⓜ] in class Thread[Ⓜ]

Página con los detalles principales de la clase **Servidor**:

PACKAGE

CLASS

USE

TREE

INDEX

HELP

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHODSEARCH

Package `ejercicio1`

Class **Servidor**

`java.lang.Object`
`ejercicio1.Servidor`

```
public class Servidor
extends Object
```

Clase / Entidad Servidor.

Author:
kzdesigner

Constructor Summary

Constructors

Constructor	Description
<code>Servidor()</code>	

Method Summary

All MethodsStatic MethodsConcrete Methods

Modifier and Type	Method	Description
static void	<code>main(String[] args)</code>	Método 'main' dedicado a construir y habilitar el servidor.

Methods inherited from class `java.lang.Object`
`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Details

Servidor

```
public Servidor()
```

Method Details

main

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

Método 'main' dedicado a construir y habilitar el servidor.

Parameters:
args - Argumentos