Evento de Discord para preparación del Examen Junio 2024 - 2025

El evento tendrá lugar el sábado 31 de mayo a las 18:00 (hora española) por Discord (enlace de descarga aquí), una aplicación gratuita que permite videoconferencias grupales, así como compartir pantalla y chat de voz. Quién les escribe será el encargado de moderar y llevar a cabo la explicación, aunque contaremos con la colaboración de otros compañeros respondiendo dudas.

La invitación del evento a la que tendrán que unirse una vez tengan la aplicación instalada es este Enlace a la clase y el evento tendrá una duración aproximada de 2 horas.

La idea de este evento es tratar de dar una explicación del proceso que habrá que realizar para llevar a cabo el examen. Se enfoca en un acercamiento práctico a la realización de los ejercicios. Se tratarán temas repaso de concurrencia en Java y profundizaremos en la creación paso a paso de un servidor HTTP.

Pasos para crear un servidor HTTP en Java:

1. **Crear clase principal ServidorHTTP**

```
public class ServidorHTTP {

    public static void main(String[] args) throws Exception {
```

2. **Crear el método main con la creación de ServerSocket y el puerto**

```
public static void main(String[] args) throws Exception {
    ServerSocket serverSocket = new ServerSocket(12349);
    System.out.println("Servidor iniciado en el puerto 12349. Esperando reservas...");
```

3. **Crear el bucle que va a escuchar peticiones y lanzar el hilo**

```
public static void main(String[] args) throws Exception {
    ServerSocket serverSocket = new ServerSocket(12349);
    System.out.println("Servidor iniciado en el puerto 12349.
    while (true) {
        Socket clientSocket = serverSocket.accept();
        new Thread(new HiloServidor(clientSocket)).start();
    }
}
```

4. **Crear clase HiloServidor que contendrá la lógica para manejar las peticiones**

```
class HiloServidor extends Thread {
    private final Socket cliente;
```

5. **Hilo servidor recibe el socket donde envía los mensajes el cliente, se lo guarda(a través del constructor)**

```
class HiloServidor extends Thread {
    private final Socket cliente;

    public HiloServidor(Socket cliente) {
        this.cliente = cliente;
    }
```

6. **La clase HiloServidor es una clase que extiende Thread o implementa Runnable por tanto utiliza su método run**

```
@Override
public void run() {
```

7. **En HiloServidor, se crean los BufferedReader y PrintWriter que serán los encargados de leer o escribir los mensajes del cliente. Siempre utilizaremos una try-with-resources o en su defecto try/catch**

```
try (BufferedReader entrada =
        new BufferedReader(new InputStreamReader(cliente.getInputStream()));
        PrintWriter salida =
                new PrintWriter(cliente.getOutputStream(), true)) {
```

8. **Empezamos tratando las peticiones y comprobamos que si la petición es nula o no corresponde a una petición de tipo GET o POST devolvemos que no se ha encontrado**

```
String peticion = entrada.readLine(); // Lee la primera línea de la petición HTTP.
if (peticion != null
        && (!peticion.startsWith("GET")
        || !peticion.startsWith("POST"))) {
```

9. **Opcional: Obtendríamos entonces la ruta a la que se dirige la petición. Puede darse el caso de que nos pidan acceso una ruta diferente al inicio /**

```
    String ruta = peticion.split(" ")[1]; // Extrae la ruta solicitada.
    StringBuilder cuerpo = new StringBuilder(); // Para almacenar el cuerpo
    String linea;
```

10. **Opcional: Obtendríamos el content-length(sólo nos sevirá en el caso de una petición POST, una petición GET no tiene cuerpo, por tanto, el content-lenght siempre será 0)**

```
String linea;

// Leer encabezados HTTP y determinar el tamaño del cuerpo.
int contentLength = 0;
while (!(linea = entrada.readLine()).isBlank()) {
    if (linea.startsWith("Content-Length: ")) {
        contentLength = Integer.parseInt(linea.substring(16));
    }
}
```

11. **Comenzamos a clasificar la petición a través de if/else si son GET o POST(si nos los pidieran también clasificaríamos la petición por la ruta o recurso al que se dirige)**

```
if (peticion.startsWith("GET")
        && (ruta.equals("/") || ruta.equals("/favicon.ico"))) {
    enviarRespuesta(salida, 200, Paginas.HTML_RESERVAS);
} else if (peticion.startsWith("POST") && contentLength > 0) {
    respuesta = procesarPostRequest(entrada, contentLength);

    if (respuesta.equals("Cantidad invalida.")) {
        enviarRespuesta(salida, 400, Paginas.HTML_ERROR);
    } else {
        enviarRespuesta(salida, 200, Paginas.HTML_RESERVAS);
    }
} else {
    enviarRespuesta(salida, 404, Paginas.HTML_NO_ENCONTRADO);
}
```

12. Según cada caso se devolverá un código de respuesta HTTP y una respuesta en HTML

```
private void enviarRespuesta(PrintWriter writer, int codigo, String contenido) {
    String statusLine;

    switch (codigo) {
        case 200 ->
            statusLine = "HTTP/1.1 200 OK";
        case 400 ->
            statusLine = "HTTP/1.1 400 Bad Request";
        case 404 ->
            statusLine = "HTTP/1.1 404 Not Found";
        default -> {
            statusLine = "HTTP/1.1 500 Internal Server Error";
            contenido = "<h1>Error desconocido</h1>";
        }
    }

    writer.println(statusLine + "\n"
            + "Content-Type: text/html; charset=UTF-8\n"
            + "Content-Length: " + contenido.length() + "\n"
            + "\n"
            + contenido);
}
```

Por último, comentaremos los recursos adicionales que deberíamos llevar al examen, así como se responderán todas las dudas que surjan.

¡¡¡¡ Les esperamos a todos !!!!