Repaso DAD 1

Syncronize se utiliza cuando tenemos hilos que comparten objetos para evitar la concurrencia
Sockets: canales de comunicación entre 2 o más procesos

- Se usa un objeto de la clase socket en cada uno de los procesos que se están comunicando (Uno distinto por programa)
- Para mandar datos se usa un flujo de escritura, pero como es tedioso, se utilizan buffers de escritura y buffers de lectura:
  - PrintWritter: Manda al bufferReader del otro extremo del socket (Para mandar de un proceso a otro)
  - BufferReader: Recibe lo introducido en el PrintWritter del otro extremo (Para recibir lo enviado por otro proceso)
  - No se puede escribir en ambos extremos a la vez, tampoco se debe poder leer en los 2 extremos a la vez ya que podemos bloquear el programa (Ambos esperan a que el otro termine de leer)
- ServerSocket: Esta clase se utiliza para un socket que quedará a la escucha como servidor, mientras que el otro socket se conectará a este como cliente.

Contenido parcial 1

1. Preguntas pequeñas de teoría 2 a 3 puntos
    1. Computación distribuida
    2. Hilos
    3. Sockets
2. Preguntas programación 1 a 2 Puntos
    1. Detectar Fallos en el código
3. Práctica
    1. Implementar un Cliente y un servidor con Sockets
    2. El profe se va a inventar un protocolo/funcionalidad y hay que ver como se va a programar (Pensar)
    3. Ver que respuesta dan el servidor a un comando de un cliente por sockets (Como el chat pero con Cases)
    4. Server Socket → While True, lanzo hilo
    5. Si un cliente tiene un listado de algo que guarda en el objeto hilo
    6. Los hilos comparten la información para estar centralizados
    7. Conexión con otros hilos a través del serverSocket

```
constructor<x,y,z>{
  this.x = x;
}
run(){
  this.x
}
```

Sugerencias/consejos del profe

- Si pide un protocolo, en el cliente debemos tener como un menú de consola o meter cosas para usar el protocolo y que el cliente mande algo al servidor para que este ejecute lo que tenga que escribir
- Construimos el servidor para el protocolo. El Servidor debe tener un Switch para que haga una acción en función a lo que reciba del cliente
- El cliente debe ser de base un cliente vacío que tenga un readln y un flush (PW.println()), una secuencia de comandos seguidos, de tal forma que para probar el servidor solo tengamos que ejecutar el cliente (no es necesario meter los comandos manualmente).
- Para cada comando o funcionalidad del servidor debemos ejecutar el cliente y meter algo por teclado, para evitar perder tiempo debemos ir metiendo todos los comandos que queramos ejecutar a machete.

```
//Estructura del server en pseudocódigo
While True
  Creamos server socket
  Acepto conexión
  Creo Hilo
  Lanzo Hilo
```

Utilizaremos una implementación similar a la del chat de clase (Multicliente)

Para la práctica (Incompleto)

```
public class ClaseAGuardar implements Serializable{
  private String campo;
  public String getCampo(){
    return campo;
  }
  public String setCampo(String campo){
    this.campo = campo;
  }
}
```

```
public class EscribirEnFichero{
  public static void main(String args[]){
    FileOutputStream fos = null;
    ObjectOutputStream oos= null;
  }
}
```

From:
https://knoppia.net/ - **Knoppia**

Permanent link:
https://knoppia.net/doku.php?id=dad:repasoparcial1

Last update: **2023/11/09 11:13**