DOCUMENTACIÓN DEL GESTOR DE MENSAJES

Contents

public void GestorMensajes(int puerto_PropioTcp, int puerto_PropioUdp): 2
public void EnviarMensaje():..... 2
public void AñadirMensajeContenedor(mensaje msg): 2
public Mensaje CogerMensajeDelContenedor(): 2
public Mensaje ComprobarContenedorDeMensajes(): 2
public void EnviaUdp(string msg, string host, string puerto):..... 3
public class RecibeTcp extends Thread{...}: 3
public void EnviaTcp(String msg, InetAddress host, int puerto): 3
public void run():..... 3
public class RecibeUdp extends Thread{...}:..... 4
ANEXO..... 4

public void GestorMensajes(int puerto_PropioTcp, int puerto_PropioUdp):

Recibe: Puerto tcp y Puerto udp del agente.
Devuelve: Nada, es una función void.
Funcionamiento: Llama al constructor de la clase que hereda, en este caso de la clase Thread. Además guarda los dos puertos pasados como parámetros en dos atributos propios de la clase: “Puerto_PropioTcp” y “Puerto_PropioUdp”.

public void EnviarMensaje():

Recibe: Nada,no hay que pasarle nada por parámetros.
Devuelve: Nada, es una función void.
Funcionamiento: Envía todos los mensajes del contenedor de mensajes. Si el contenedor está vacío se duerme y luego vuelve a mirar. Esta función está siempre en ejecución (while true) y empieza a funcionar cuando se inicializa el gestor de mensajes. En caso de haber mensaje en la cola, comprobará el protocolo necesario para enviarlo y llamará a EnviaTcp o EnviaUdp.

public void AñadirMensajeContenedor(mensaje msg):

Recibe: Un objeto mensaje, el cual se añadirá al contenedor.
Devuelve: Nada, es una función void.
Funcionamiento: Mete el mensaje pasado por parámetro al contenedor de mensaje usando mutual exclusion.

public Mensaje CogerMensajeDelContenedor():

Recibe: No recibe nada por parámetro.
Devuelve: Un objeto de la clase Mensaje.
Funcionamiento: Con la función pop() se extrae un mensaje del contenedor de mensajes recibidos. Se utiliza mutual exclusion para impedir que dos hilos accedan a la vez al contenedor. Esta función asume que al menos hay un mensaje que extraer.

public Mensaje ComprobarContenedorDeMensajes():

Recibe: No recibe nada por parámetro.
Devuelve: Un boolean.
Funcionamiento: Con la función isEmpty() del **contenedor_de_mensajes_recibidos** (**LinkedList<Mensaje>**)**,** se comprueba si el contenedor está vacío o no. Se utiliza exclusión mutua para impedir que dos hilos accedan a la vez al contenedor, evitando un resultado erróneo si se añade o se retira un mensaje a la vez que se comprueba si está vacío el

contenedor. Esta función devuelve **TRUE** si el contenedor está vacío, es decir, no hay mensajes nuevos, y **FALSE** en caso contrario: si hay mensajes en el contenedor.

public void EnviaUdp(string msg, string host, string puerto):

Recibe: Un mensaje en forma de string, el host en forma de string, y el puerto en forma de string.
Devuelve: Nada
Funcionamiento: Manda el mensaje utilizando el protocolo UDP al host y puerto que se pasa por parámetro.

public class RecibeTcp extends Thread{...}:

Recibe: No recibe nada por parámetro. Extiende de la clase Thread.
Devuelve: Nada, es una clase.
Funcionamiento: Crea un socket vinculado al "Puerto_PropioTcp" de la clase GestorMensajes. Se queda permanentemente en escucha esperando un mensaje enviado mediante ese socket. Obtiene el flujo de entrada y lee el objeto del stream. Llamamos a la función "ProcesaMensaje(mensaje)", la cual se encargará de decidir que se hace con dicho mensaje leído.
Tratamiento de excepciones: Si llegamos a un error, imprimimos la excepción correspondiente, seguidamente llamamos de igual forma a la función "ProcesaMensaje(mensaje)" con dicho mensaje de error.

public void EnviaTcp(String msg, InetAddress host, int puerto):

Recibe: un dato de tipo String, es el mensaje que se envía, una dirección IP de tipo inetAddress y un int que indica el puerto a donde se envía.
Devuelve: no devuelve nada.
Funcionamiento: La función crea un nuevo socket, con la dirección y el puerto que llegan a la función. A continuación, crea un flujo de datos de salida. A través del flujo de datos envía el dato almacenado en el String que le llega a la función. Una vez termina de enviar el String, cierra el flujo de datos y acto seguido cierra el socket. Al final de la ejecución y si todo ha ido bien, escribe por pantalla.
Tratamiento de excepciones: Si llegamos a un error, imprimimos primero un mensaje genérico de error junto con la excepción, genera un mensaje para añadir al contenedor de mensajes y acto seguido añade el mensaje creado.

public void run():

Recibe: no recibe nada por parámetro.
Devuelve: no devuelve nada.

Funcionamiento: crea e inicia dos hilos para recibir mensajes, uno por UDP o otro por TCP.

public class RecibeUdp extends Thread{...}:

Recibe: No recibe nada por parámetro. Extiende de la clase Thread.
Devuelve: Nada, es una clase.
Funcionamiento: primero, crea un DatagramSocket vinculado al “Puerto_PropioUdp” de la clase GestorMensajes. Después, obtiene el flujo de entrada y se queda a la escucha hasta recibir el mensaje. Por último, llamamos a la función “ProcesaMensaje(mensaje.toString())”, para decidir qué hacer con el mensaje leído.

ANEXO

La documentación que aparece a continuación es generada automáticamente por JavaDoc, y es más completa a la par que más compleja.

Class GestorMensajes

java.lang.Object
    java.lang.Thread
        GestorMensajes

All Implemented Interfaces:

Runnable

```
public class GestorMensajes
extends Thread
```

GestorMensajes es la clase que hemos visto necesaria crear para poder gestionar todos los mensajes, tanto recibidos como enviados.
Hemos visto necesario utilizar una clase aparte en vez de utilizar un método.
Si no, no podríamos hacerlo que queremos hacer con esta clase.
Esta clase es una forma de abstraer la lógica de comunicación y mensajería.

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

Thread.State , Thread.UncaughtExceptionHandler

Field Summary

Fields

Modifier and Type	Field	Description
LinkedList <Mensaje>	contenedor_de_mensajes_a_enviar	

LinkedList <Mensaje>	contenedor_de_mensajes_recibidos
int	Puerto_PropioTcp
int	Puerto_PropioUdp

Fields inherited from class java.lang.Thread

MAX_PRIORITY , MIN_PRIORITY , NORM_PRIORITY

Constructor Summary

Constructors

Constructor	Description
GestorMensajes(int puerto_PropioTcp, int puerto_PropioUdp)	**Constructor de la clase GestorMensajes** Llama al constructor de la clase que hereda, en este caso de la clase Thread. Además guarda los dos puertos pasados como parámetros en dos atributos propios de la clase: “Puerto_PropioTcp” y “Puerto_PropioUdp”.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	AñadirMensajeContenedor(Mensaje msg)	Añade el (Mensaje) msg al (LinkedList) contenedor_de_mensajes_recibidos. Esta operación se realiza con exclusión mutua, para que no haya más de un hilo a la vez realizándola.

Mensaje	CogerMensajeDelContenedor()	Con la función pop() se extrae un mensaje del contenedor de mensajes recibidos. Se utiliza mutual exclusion para impedir que dos hilos accedan a la vez al contenedor. Esta función asume que hay como mínimo un mensaje que extraer.
boolean	ComprobarContenedorDeMensajes()	Con la función isEmpty() del (LinkedList) contenedor_de_mensajes_recibidos, se comprueba si el contenedor está vacío o no. Se utiliza exclusión mutua para impedir que dos hilos accedan a la vez al contenedor, evitando un resultado erróneo si se añade o se retira un mensaje a la vez que se comprueba si está vacío el contenedor.
void	EnviarMensaje()	**EnviarMensaje()** envia todos los mensajes de la cola.
void	EnviaTcp(String msg, String host, String puerto)	La función crea un nuevo socket, con la dirección y el puerto que llegan a la función. A continuación, crea un flujo de datos de salida. A través del flujo de datos envía el dato almacenado en el String que le llega a la función. Una vez termina de enviar el String, cierra el flujo de datos y acto seguido cierra el socket. Al final de la ejecución y si no ha habido fallos, escribe por pantalla.
void	EnviaUdp(String msg, String host, String puerto)	Manda el mensaje utilizando el protocolo UDP la host y puerto pasados por parámetro.
HashMap <String ,Object >	generaCab(HashMap <String ,Object > cab)	
void	ProcesaMensaje(String xml)	

void	run()	Crea e inicia dos hilos, uno TCP (clase **RecibeTcp**) y el otro UDP (clase **RecibeUdp**), para recibir mensajes.
String	TransformarMensaje(Mensaje mA)	ENTRADA: Instancia de mensajeAEnviar* FUNCION: Vamos a sacar los datos necesarios de mensajeAEnviar para crear el XML y lo transformaremos a String para que nuestros compañeros puedan enviarlo

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 , join , join , join , onSpinWait , resume , setContextClassLoader , setDaemon , setDefaultUncaughtExceptionHandler , setName , setPriority , setUncaughtExceptionHandler , sleep , sleep , start , stop , suspend , toString , yield

Methods inherited from class java.lang.Object

equals , finalize , getClass , hashCode , notify , notifyAll , wait , wait , wait

Field Details

contenedor_de_mensajes_a_enviar

```
public LinkedList <Mensaje> contenedor_de_mensajes_a_enviar
```

contenedor_de_mensajes_recibidos

```
public LinkedList <Mensaje> contenedor_de_mensajes_recibidos
```

Puerto_PropioTcp

```
public int Puerto_PropioTcp
```

Puerto_PropioUdp

```
public int Puerto_PropioUdp
```

Constructor Details

GestorMensajes

```
public GestorMensajes(int puerto_PropioTcp,
                      int puerto_PropioUdp)
```

Constructor de la clase GestorMensajes Llama al constructor de la clase que hereda, en este caso de la clase Thread.
Además guarda los dos puertos pasados como parámetros en dos atributos propios de la clase: “Puerto_PropioTcp” y “Puerto_PropioUdp”.

Parameters:

`puerto_PropioTcp` - el valor del puerto TCP del agente para la instancia de gestor de mensajes

`puerto_PropioUdp` - el valor del puerto UDP del agente para la instancia de gestor de mensajes

Method Details

run

```
public void run()
```

Crea e inicia dos hilos, uno TCP (clase **RecibeTcp**) y el otro UDP (clase **RecibeUdp**), para recibir mensajes.

Specified by:

`run` in interface `Runnable`

Overrides:

`run` in class `Thread`

EnviarMensaje

```
public void EnviarMensaje()
                  throws InterruptedException ,
                         ParserConfigurationException ,
                         IOException ,
                         SAXException ,
                         jdk.internal.org.xml.sax.SAXException,
                         TransformerException
```

EnviarMensaje() envia todos los mensajes de la cola. Si la cola está vacía se duerme y luego vuelve a mirar.

Esta función está siempre en ejecución (while true) y empieza a funcionar cuando se inicializa el gestor de mensajes.

En caso de haber mensaje en la cola, comprobará el protocolo necesario para enviarlo y llamará a EnviaTcp o EnviaUdp.

Throws:

`InterruptedException`

`ParserConfigurationException`

`IOException`

`SAXException`

```
jdk.internal.org.xml.sax.SAXException

TransformerException
```

AñadirMensajeContenedor

```
public void AñadirMensajeContenedor(Mensaje msg)
```

Añade el (Mensaje) msg al (LinkedList) contenedor_de_mensajes_recibidos.
Esta operación se realiza con exclusión mutua, para que no haya más de un hilo a la vez realizándola.

Parameters:

`msg` -

CogerMensajeDelContenedor

```
public Mensaje CogerMensajeDelContenedor()
```

Con la función pop() se extrae un mensaje del contenedor de mensajes recibidos.
Se utiliza mutual exclusion para impedir que dos hilos accedan a la vez al contenedor.
Esta función asume que hay como mínimo un mensaje que extraer. Por ello, se debe llamar a ComprobarContenedorDeMensajes() primero.

Returns:

Objeto de la clase Mensaje referente al último mensaje añadido a la pila (LinkedList) contenedor_de_mensajes_recibidos.

ComprobarContenedorDeMensajes

```
public boolean ComprobarContenedorDeMensajes()
```

Con la función isEmpty() del (LinkedList) contenedor_de_mensajes_recibidos, se comprueba si el contenedor está vacío o no.
Se utiliza exclusión mutua para impedir que dos hilos accedan a la vez al contenedor, evitando un resultado erróneo si se añade o se retira un mensaje a la vez que se comprueba si está vacío el contenedor.

Returns:

True => El contenedor está vacío, no hay mensajes nuevos

False => El contenedor contiene como mínimo un mensaje.

EnviaTcp

```
public void EnviaTcp(String  msg,
                     String  host,
                     String  puerto)
              throws ParserConfigurationException ,
                     IOException ,
                     SAXException ,
                     jdk.internal.org.xml.sax.SAXException
```

La función crea un nuevo socket, con la dirección y el puerto que llegan a la función.
A continuación, crea un flujo de datos de salida.
A través del flujo de datos envía el dato almacenado en el String que le llega a la función.
Una vez termina de enviar el String, cierra el flujo de datos y acto seguido cierra el socket.

Al final de la ejecución y si no ha habido fallos, escribe por pantalla.

Parameters:

`msg` - El mensaje a enviar, de tipo String

`host` - la dirección IP a la que enviar el mensaje, de tipo InetAddress

`puerto` - el puerto de escucha TCP del destinatario

Throws:

`ParserConfigurationException`

`IOException`

`SAXException`

`jdk.internal.org.xml.sax.SAXException` -

Si detectamos un error, imprimimos un mensaje genérico de error junto con la excepción.
Después, genera un mensaje para añadir al contenedor de mensajes y por último, añade el mensaje creado.

EnviaUdp

```
public void EnviaUdp(String  msg,
                     String  host,
                     String  puerto)
```

Manda el mensaje utilizando el protocolo UDP la host y puerto pasados por parámetro.

Parameters:

`msg` - El mensaje a enviar, de tipo String

`host` - la dirección IP a la que enviar el mensaje, también de tipo String

`puerto` - el puerto de escucha TCP, en formato String

generaCab

```
public HashMap <String ,Object > generaCab(HashMap <String ,Object > cab)
```

Parameters:

`cab` - , en la que se espera que se incluyan los siguiente parametros:

1. ip_Receptor:InetAdress
2. puerto_Receptor:int
3. id_Receptor:String
4. tipoMensaje:String
5. protocolo:String
6. id_mensaje:String

Returns:

cabecera completa, con la Ip del emisor, puerto, id, hora de generacion y los parametros pasados, ademas dek orden correcto

ProcesaMensaje

```
public void ProcesaMensaje(String  xml)
```

```
                    throws ParserConfigurationException ,
                           IOException ,
                           SAXException ,
                           jdk.internal.org.xml.sax.SAXException
```

Throws:

ParserConfigurationException

IOException

SAXException

jdk.internal.org.xml.sax.SAXException

TransformarMensaje

```
public String  TransformarMensaje(Mensaje mA)
                          throws IOException ,
                                 SAXException ,
                                 ParserConfigurationException ,
                                 TransformerException
```

ENTRADA: Instancia de mensajeAEnviar* FUNCION: Vamos a sacar los datos necesarios de mensajeAEnviar para crear el XML y lo transformaremos a String para que nuestros compañeros puedan enviarlo

Throws:

IOException

SAXException

ParserConfigurationException

TransformerException

Class RecibeTcp

java.lang.Object
    java.lang.Thread
        RecibeTcp

All Implemented Interfaces:

`Runnable`

```
public class RecibeTcp
extends Thread
```

Esta clase se ha creado con el motivo de separar la lógica de recepción de mensajes TCP

Crea un socket vinculado al “Puerto_PropioTcp” de la clase GestorMensajes.
Se queda permanentemente en escucha esperando un mensaje enviado mediante ese socket.
Obtiene el flujo de entrada y lee el objeto del stream. Llamamos a la función “ProcesaMensaje(mensaje)”, la cual se encargará de decidir que se hace con dicho mensaje leído.

Tratamiento de excepciones Si llegamos a un error, imprimimos la excepción correspondiente, seguidamente llamamos de igual forma a la función “ProcesaMensaje(mensaje)” con dicho mensaje de error

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

`Thread.State` , `Thread.UncaughtExceptionHandler`

Field Summary

Fields inherited from class java.lang.Thread

MAX_PRIORITY , MIN_PRIORITY , NORM_PRIORITY

Constructor Summary

Constructors

Constructor	Description
RecibeTcp()	

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	run()	Método que se va a llamar al iniciar el hilo de esta clase

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 , join , join , join , onSpinWait , resume , setContextClassLoader , setDaemon , setDefaultUncaughtExceptionHandler , setName , setPriority , setUncaughtExceptionHandler , sleep , sleep , start , stop , suspend , toString , yield

Methods inherited from class java.lang.Object

equals , finalize , getClass , hashCode , notify , notifyAll , wait , wait , wait

Constructor Details

RecibeTcp

```
public RecibeTcp()
```

Method Details

run

```
public void run()
```

Método que se va a llamar al iniciar el hilo de esta clase

Specified by:

`run` in interface `Runnable`

Overrides:

`run` in class `Thread`

Class RecibeUdp

java.lang.Object
    java.lang.Thread
        RecibeUdp

All Implemented Interfaces:

Runnable

```
public class RecibeUdp
extends Thread
```

Esta clase se ha creado con el motivo de separar la lógica de recepción de mensajes UDP

Primero, crea un DatagramSocket vinculado al “Puerto_PropioUdp” de la clase GestorMensajes.
Después, obtiene el flujo de entrada y se queda a la escucha hasta recibir el mensaje.
Por último, llamamos a la función “ProcesaMensaje(mensaje.toString())”, para decidir qué hacer con el mensaje leído

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

Thread.State , Thread.UncaughtExceptionHandler

Field Summary

Fields inherited from class java.lang.Thread

MAX_PRIORITY , MIN_PRIORITY , NORM_PRIORITY

Constructor Summary

Constructors

Constructor	Description
`RecibeUdp()`	

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	`run()`	Método que se va a llamar al iniciar el hilo de esta clase

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 , join , join , join , onSpinWait , resume , setContextClassLoader , setDaemon , setDefaultUncaughtExceptionHandler , setName , setPriority , setUncaughtExceptionHandler , sleep , sleep , start , stop , suspend , toString , yield

Methods inherited from class java.lang.Object

equals , finalize , getClass , hashCode , notify , notifyAll , wait , wait , wait

Constructor Details

RecibeUdp

```
public RecibeUdp()
```

Method Details

run

```
public void run()
```

Método que se va a llamar al iniciar el hilo de esta clase

Specified by:

`run` in interface `Runnable`

Overrides:

`run` in class `Thread`