SimpleServer

Thursday, 27. April 2017

Johannes Posch

Abstract

In this document the library SimpleServer is described.

Contents

1 The Library

The library SimpleServer provides simple way to create Servers and to manage them. The library should be used to create server applications. It is not the most powerful library, but it's easier to achieve big results. This system is written in Java and is also thought to be used with it.
The library consists of three different packages:

- simpleServer.exception
- simpleServer.server
- simpleServer.util

These packages hold different classes:

- simpleServer.exception: This package contains the customized exceptions for this project. This package contains three classes:
  1. IllegalPortException goto
  2. NoSuchServerException goto
  3. RunningServerException goto
- simpleServer.server: In this package the main server-classes are defined. This package contains three classes:
  1. Server goto
  2. InsecureServer goto
  3. SecureServer goto
- simpleServer.util: This package contains configuration classes for the servers and also elements to support handling of multiple servers and networking services. This package contains seven classes:
  1. Constant goto
  2. Database_base goto
  3. Handler goto
  4. SecureHandler goto
  5. NetworkService goto
  6. SecureNetworkService goto
  7. Ports goto
  8. ServerControl goto

1.1 Requirement for SSL(SecureServer)

SecureServer requires some configuration work to be done. To use this server 4 application-variables have to be set:

1. javax.net.ssl.trustStore
2. javax.net.ssl.trustStorePassword
3. javax.net.ssl.keyStore

4. javax.net.ssl.keyStorePassword

The Trust-store and the Key-store need to be generated before. There are two main ways to set these variables:

1. On command-line(cli), during the program start, following parameters have to be added:

   ```
   -Djavax.net.ssl.keyStore="C:\...\SSLKeyStore"
   -Djavax.net.ssl.keyStorePassword=THE_KEYSTORE_PASSWORD
   -Djavax.net.ssl.trustStore="C:\...\SSLTrustStore"
   -Djavax.net.ssl.trustStorePassword=THE_TRUSTSTORE_PASSWORD
   ```

2. In the application following function set's a variable:

   ```
   System.setProperty("javax.net.ssl.trustStore","C:\...\SSLTrustStore");
   System.setProperty("javax.net.ssl.trustStorePassword","THE_TRUSTSTORE_PASSWORD")
       ;
   System.setProperty("javax.net.ssl.keyStore","C:\...\SSLKeyStore");
   System.setProperty("javax.net.ssl.keyStorePassword","THE_KEYSTORE_PASSWORD");
   ```

   These lines have to be added at the beginning of the program.

It's necessary to create/become those files. A good explanation to do so is on the webpage: https://docs.continuent.com/tungsten-replicator-4.0/deployment-ssl-stores.html#deployment-ssl-stores-own

1.2 Default-Constants

The library contains some default constants. These are defined in the Constants class and can also be changed using their functions. The class is static and also their function's can be called statically. The default values are:

- JDBC_DRIVER [String]: com.mysql.jdbc.Driver
- minimalServerPort [int]: 5000
- maximalServerPort [int]: 10000

1.3 How to's

1.3.1 How to change server-port borders

The library uses default values for the server-ports. These can be changed by using class Ports. These new set borders can then be also accessed by class Constant.

```
Ports portObject = Ports.getInstance();
portObject.setMinServerPort(0);
portObject.setMaxServerPort(65535);
```

1.3.2 How to change database driver

Updating the database-driver-string is done by calling the corresponding function on class Constant.

```
Constant.updateDatabaseDriver("com.mysql.jdbc.Driver");
```

2 Package-simpleServer.exception

The package simpleServer.exception contains at this point three exceptions. These are customized for this project.
The exceptions are:

1. IllegalPortException
2. NoSuchServerException
3. RunningServerException

2.1 IllegalPortException

This exception signalizes that the given port is invalid and that it cannot be used to work with a server. The main reason, why this exception exists, is a specification in the library. Per default the servers can only have port-numbers between 5000 and 10000. This is because there are only a few already existing ports on the system. This range can be changed(see: How to change server-port borders)

2.2 NoSuchServerException

This exception signalizes that the requested server does not exist in ServerControl.

2.3 RunningServerException

This exception indicates, that the server is running, on which a configuration action should be performed.

3 Class: Server

This class is used as base-class for the classes InsecureServer and SecureServer. It's recommended not to use this class. This class is abstract so it cannot be instantiated. This class should only be used to extend from.

The class take's from it's constructor the server-name and a port to create the server on. To start(...) method is used to start the server. The server start's it's socket and then calls the method start_Handler_service(...) which needs to be implemented. At a server stop the server calls stop_Handler_service(...) and closes the socket.

3.1 public Server(final String name, final int port)

This is the constructor, which defines the object in first instance. The constructor takes the server-name as parameter. This name can be used to display the server and it's identity for persons.

3.1.1 Parameters

The function takes 2 parameters:

- name [string]: the name for the new server object
- port [int]: the port, for the server to start on

3.1.2 exceptions

The function throws following exceptions:

- NullPointerException
- IllegalPortException

3.2 public boolean start()

This function starts the server. It creates the socket and starts it. This function then calls the function start_Handler_service(...).

3.2.1 exceptions

The function throws following exceptions:

- RunningServerException

3.2.2 Returnvalue

The function returns a boolean value:

- true: the start of the server was successful
- false: the start was not successful (The most recent reason would be because the port is not available or start_Handler_service(...) returned false)

3.3 public boolean stop()

This function calls the procedure stop_Handler_service(...). After this function has stopped successful the server's socket gets closed.

3.3.1 exceptions

The function throws following exceptions:

- IOException: this exception occures on an error in socket close

3.3.2 Returnvalue

The function returns a boolean value:

- true: the stop of the server was successful
- false: the server could not be stopped(either because the server is not running or the stop_Handler_service returned false)

3.4 protected abstract boolean start_Handler_service()

This function is an unimplemented and get's called on a server start. The function has to be implemented in the server class, which extends this class.

3.4.1 exceptions

This function should not throw an error. The implementation should catch every exception itself.

3.4.2 Returnvalue

The function's return-value should define a successful or unsuccessful start.

3.5 protected abstract boolean stop_Handler_service()

This function is an unimplemented and get's called on a server stop. The function has to be implemented in the server class, which extends this class.

3.5.1 exceptions

This function should not throw an error. The implementation should catch every exception itself.

3.5.2 Returnvalue

The function's return-value should define a successful or unsuccessful stop.

3.6 public void setPort(final int port)

This function set's the port of the server. For this action the server has to be stopped.

3.6.1 Parameters

The function takes 1 parameter:

1. port [int]: The new port for the server

3.6.2 exceptions

The function throws following exceptions:

- IllegalPortException
- RunningServerException

3.7 public int getPort()

This function returns the actual configured port of the server.

3.7.1 Returnvalue

This function returns the actual configured port of the server as int.

3.8 public boolean isRunning()

This function returns if the server is actual running.

3.8.1 Returnvalue

This function returns if the server is actual running as boolean.

3.9 protected void setRunning(final boolean running)

This function is to set the running variable of the server. It is not recommended to use this function in any case.

3.9.1 Parameters

The function takes 1 parameter:

- running: This is the new value of the variable running

3.10 public String getName()

This function returns the given name of the server.

3.10.1 Returnvalue

The function returns the given name of the server as String.

4 Class: InsecureServer

This class is to be used for servers with a plain socket. It is a specified version of the class Server and extends also from that class. This class is abstract so it cannot be instantiated. This class should be used to extend from.

4.1 protected ServerSocket getSSocket()

This function returns the ServerSocket of the server.

4.1.1 Returnvalue

The function returns an ServerSocket object of the actual server. NULL will be returned if the server is not running.

4.2 public InsecureServer(final String name, final int port)

see Server::Server

4.3 public boolean start()

see Server::start

4.4 public boolean stop()

see Server::stop

4.5 protected abstract boolean start_Handler_service()

see Server::start_Handler_Service

4.6 protected abstract boolean stop_Handler_service()

see Server::stop_Handler_Service

4.7 public void setPort(final int port)

see Server::setPort

4.8 public int getPort()

see Server::getPort

4.9 public boolean isRunning()

see Server::isRunning

4.10 protected void setRunning(final boolean running)

see Server::setRunning

4.11 public String getName()

see Server::getName

5 Class: SecureServer

This class is to be used for servers with a secured ssl socket. It is a specified version of the class Server and extends also from that class. This class is abstract so it cannot be instantiated. This class should be used to extend from.

5.1 protected SSLServerSocket getSSocket()

This function returns the SSLServerSocket of the server.

5.1.1 Returnvalue

The function returns an SSLServerSocket object of the actual server. NULL will be returned if the server is not running.

5.2 public InsecureServer(final String name, final int port)

see Server::Server

5.3 public boolean start()

see Server::start

5.4 public boolean stop()

see Server::stop

5.5 protected abstract boolean start_Handler_service()

see Server::start_Handler_Service

5.6 protected abstract boolean stop_Handler_service()

see Server::stop_Handler_Service

5.7 public void setPort(final int port)

see Server::setPort

5.8 public int getPort()

see Server::getPort

5.9 public boolean isRunning()

see Server::isRunning

5.10 protected void setRunning(final boolean running)

see Server::setRunning

5.11 public String getName()

see Server::getName

6 Class: Constant

This class is a final class. In this class some constants are defined to easier control the system. This constants can be updated at every time.

6.1 public static int getMinServerPort()

This function returns the smallest server-port to configure.

6.1.1 Returnvalue

The function returns the smallest available port for the system.

6.2 public static int getMaxServerPort()

This function returns the highest server-port to configure.

6.2.1 Returnvalue

The function returns the highest available port for the system.

6.3 public static void updateDatabaseDriver(final String driver)

This function updates the database driver variable, which can be used to have a system-wide unified driver.

6.3.1 Parameters

The function takes 1 parameter:

1. driver [String]: The new driver-string

6.4 public static String getDatabaseDriver()

This function returns the configured database-driver string.

6.4.1 Returnvalue

The function returns the highest available port for the system.

7 Class: Database_base

This class provides a base class for database connection classes. The functions in the class are used to open and close connections to a database for any subclass.

7.1 Database_base(final String db_url)

The constructor takes the database string to connecto to the database.

7.1.1 Parameters

The function takes 1 parameter:

1. db_url [String]: The url to the database

7.2 public void openConnection(final String USER, final String PASS)

This function opens a connection to the desired database with the given credentials.

7.2.1 Parameters

The function takes 2 parameters:

1. USER [String]: The user to communicate with the database
2. PASS [String]: The password for the database user

7.2.2 exceptions

The function throws following exceptions:

- ClassNotFoundException
- SQLException
- DatabaseURLNotSetException

7.3 public void close()

This function closes the connection to the database.

7.3.1 exceptions

The function throws following exceptions:

- SQLException

7.4 protected Connection getConnection()

This function returns the connection object for this server.

7.4.1 Returnvalue

The connection object to the database. If the connection was not opened null will be returned.

8 Class: Handler

This class is to use the callback for the class NetworkService to call the handler defined for each NetworkService. The class implements the interface Runnable to be used in a thread.

8.1 public Handler (final Socket sock, final Consumer<Socket> handler)

The constructor assigns the values to it’s objects.

8.1.1 Parameters

The function takes 2 parameters:

- sock [Socket]: This is the socket-object given from the Network-Handler.
- handler [Consumer<Socket>]: This is the callback function. The function needs to take a Socket object and must not return a value.

8.1.2 exceptions

The function throws following exceptions:

- NullPointerException

8.2 public void run()

This function calls the handler function, defined by the constructor.

9 Class: SecureHandler

This class does exactly the same as Handler but with a SSLSocket.

9.1 public Handler (final SSLSocket sock, final Consumer<Socket> handler)

The constructor assigns the values to it's objects.

9.1.1 Parameters

The function takes 2 parameters:

- sock [SSLSocket]: This is the socket-object given from the Network-Handler.
- handler [Consumer<Socket>]: This is the callback function. The function needs to take a Socket object and must not return a value.

9.1.2 exceptions

The function throws following exceptions:

- NullPointerException

9.2 public void run()

see Handler::run

10 Class: NetworkService

The NetworkService class's aim is to simplify the multi-threading for all the incoming connections to the server. This class extends from the Thread class. In order to start it call the function start(...). The class uses the class Handler to call the handler function. There are special ways to use this class (see: ——————————————————————————————————-)

10.1 public NetworkService(final ServerSocket serverSocket, final Consumer<Socket> onRequest)

This constructor is used to give NetworkService the ServerSocket to listen from and also the callback function for each new connection.

10.1.1 Parameters

The function takes 2 parameters:

- serverSocket [ServerSocket]: This ServerSocket will be listened to, to accept connections from.
- onRequest [Consumer<Socket>]: This is the callback function. The function needs to take a Socket object and must not return a value.

10.2 public void run()

This is the thread run function. The function wait's until it receives a new connection on the serverSocket. On each new connection the function starts a thread with the given Handler function.

11 Class: SecureNetworkService

This class does exactly the same as NetworkService but with a SSLServerSocket.

11.1 public NetworkService(final SSLServerSocket serverSocket, final Consumer<Socket> onRequest)

This constructor is used to give NetworkService the ServerSocket to listen from and also the callback function for each new connection.

11.1.1 Parameters

The function takes 2 parameters:

- serverSocket [SSLServerSocket]: This ServerSocket will be listened to, to accept connections from.
- onRequest [Consumer<Socket>]: This is the callback function. The function needs to take a Socket object and must not return a value.

11.2 public void run()

see NetworkService::run

12 Class: Ports

This class is programmed using the Singleton pattern. In this class the port-borders are defined. These borders will be used to configure the servers.

12.1 private Ports()

This class's constructor is private so that nobody else can create a instance of this class. The constructor set's the default values for the port configuration.

12.2 public static Ports getInstance()

This function is necessary for the singleton pattern. The function returns an object of the class.

12.2.1 Returnvalue

The function returns an object of type Ports.

12.3 public int getMinServerPort()

This function returns the smallest server-port to configure.

12.3.1 Returnvalue

The function returns the smallest available port for the system.

12.4 public static int getMaxServerPort()

This function returns the highest server-port to configure.

12.4.1 Returnvalue

The function returns the highest available port for the system.

12.5 public void setMinServerPort(final int newPort)

This function is used to set the smallest server-port for configuration.

12.5.1 Parameters

The function takes 1 parameter:

- newPort [int]: The new smallest server-port.

12.5.2 exceptions

The function throws following exceptions:

- IllegalPortException In this case the meaning is, that the port is completely out of range (0 - 65535).

12.5.3 Returnvalue

The function returns the smallest available port for the system.

12.6 public static int getMaxServerPort()

This function is used to set the highest server-port for configuration.

12.6.1 Parameters

The function takes 1 parameter:

- newPort [int]: The new highest server-port.

12.6.2 exceptions

The function throws following exceptions:

- IllegalPortException In this case the meaning is, that the port is completely out of range (0 - 65535).

12.6.3 Returnvalue

The function returns the smallest available port for the system.

13 Class: ServerControl

This class's aim is to help managing all the server objects. All servers can be added to this control and then be operated.

13.1 public ServerControl()

The constructor initializes the ArrayList inside the object.

13.2 public boolean addServer(final Server srv)

This function add's a server to the List of controlled elements.

13.2.1 Parameters

The function takes 1 parameter:

- srv [Server]: The server object to add.

13.2.2 exceptions

The function throws following exceptions:

- NullPointerException

13.2.3 Returnvalue

The function returns true if the add process was done successfully.

13.3 public boolean removeServer(final Server srv)

This function remove's a server to the List of controlled elements.

13.3.1 Parameters

The function takes 1 parameter:

- srv [Server]: The server object to remove.

13.3.2 exceptions

The function throws following exceptions:

- NullPointerException

13.3.3 Returnvalue

The function returns true if the remove process was done successfully. And false, if the server does not exist or if the server is running.

13.4 public boolean removeServer(final int index)

This function does a lookup in the serverlist at position index and then calls the function removeServer(final Server srv)

13.4.1 Parameters

The function takes 1 parameter:

- index [int]: The index of the server object inside the container.

13.4.2 exceptions

The function throws following exceptions:

- NullPointerException

13.4.3 Returnvalue

The function returns true if the remove process was done successfully. And false, if the server does not exist or if the server is running.

13.5 public void printServerInformation()

This function prints the information whether the server is running or not and the Port on the commandline.

13.6 public void startAllServers()

This function calls for every element inside it's container the function startSpecificServer(final int index).

13.6.1 exceptions

The function throws following exceptions:

- IOException: This mostly means, that a port is already in use.

13.7 public void stopAllServers()

This function calls for every element inside it's container the function stopSpecificServer(final int index).

13.7.1 exceptions

The function throws following exceptions:

- IOException

13.8 public void startSpecificServer(final String name)

This function calls for a specific element with the given name inside it's container the function startSpecificServer(final int index).

13.8.1 Parameters

The function takes 1 parameter:

- name [String]: The name of the server object inside the container.

13.8.2 exceptions

The function throws following exceptions:

- IOException: This mostly means, that a port is already in use.
- NoSuchServerException

13.9 public void startSpecificServer(final int index)

This function starts a specific server with the given index.

13.9.1 Parameters

The function takes 1 parameter:

- index [int]: The index of the server object inside the container.

13.9.2 exceptions

The function throws following exceptions:

- IOException: This mostly means, that a port is already in use.
- IndexOutOfBoundsException

13.10 public void stopSpecificServer(final String name)

This function calls for a specific element with the given name inside it's container the function stopSpecificServer(final int index).

13.10.1 Parameters

The function takes 1 parameter:

- name [String]: The name of the server object inside the container.

13.10.2 exceptions

The function throws following exceptions:

- IOException: This mostly means, that a port is already in use.
- NoSuchServerException

13.11 public void stopSpecificServer(final int index)

This function stops a specific server with the given index.

13.11.1 Parameters

The function takes 1 parameter:

- index [int]: The index of the server object inside the container.

13.11.2 exceptions

The function throws following exceptions:

- IOException: This mostly means, that a port is already in use.
- IndexOutOfBoundsException

13.12 public String[] getServerNameList()

This function returns a String array with all server names managed by ServerControl.

13.12.1 Returnvalue

An array with all server-names in the ServerControl container.

13.13 public Server getServer(final String name)

This function returns the first server object to a server with the given name.

13.13.1 Parameters

The function takes 1 parameter:

- name [String]: The name of the server object inside the container.

13.13.2 exceptions

The function throws following exceptions:

- NoSuchServerException

13.13.3 Returnvalue

The function returns a server object on success or null on failure.

13.14 public Server getServer(final int index)

This function returns the first server object to a server with the ServerControl container index.

13.14.1 Parameters

The function takes 1 parameter:

- index [int]: The index of the server object inside the container.

13.14.2 exceptions

The function throws following exceptions:

- IndexOutOfBoundsException

13.14.3 Returnvalue

The function returns a server object on success or null on failure.

13.15 public int getServerCount()

This function returns how many servers ServerControl contains.

13.15.1 Returnvalue

The function returns the amount of servers in ServerControl container.