HIGH SCHOOL OF TECHNOLOGY ESSAOUIRA

MARRAKECH

جامعة القاضي عياض
UNIVERSITÉ CADI AYYAD

Report
socket : Course examples - Multi-Threading - Chatting App.

Realized by :
Mr. Bella Abdelouahab

Supervised By :
Pr. Fahd Karami

27 février 2025

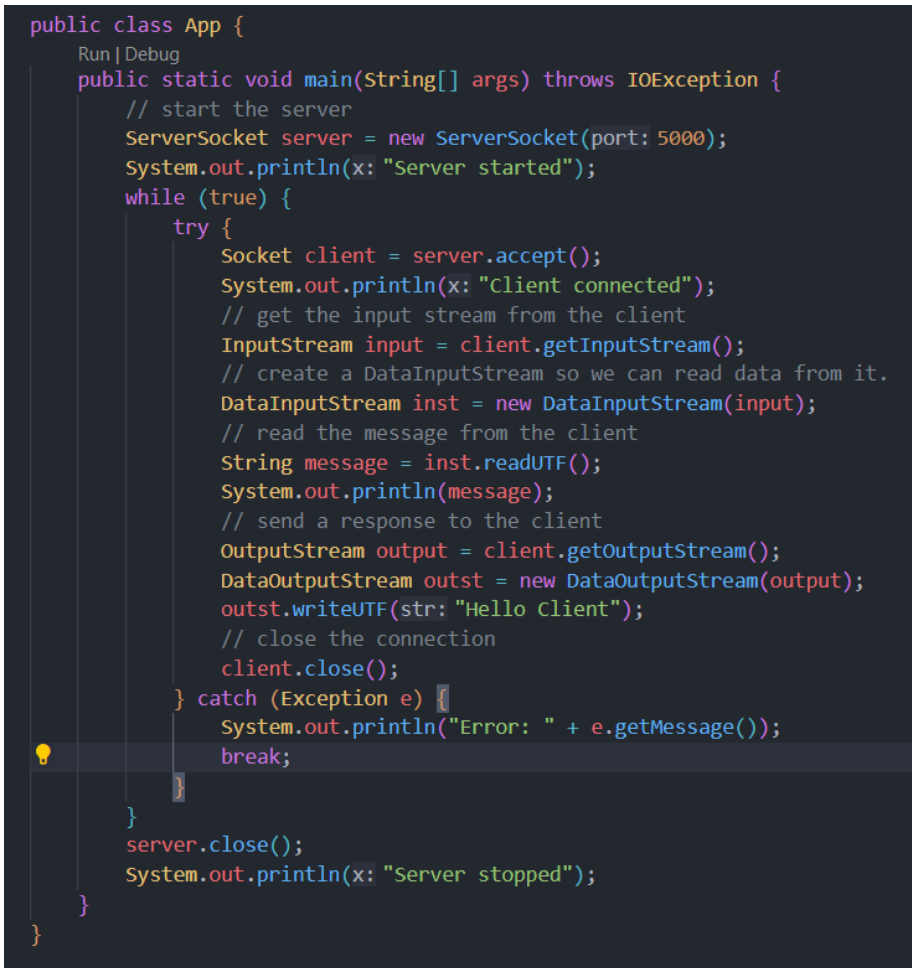
Table des matières

Table des figures

Chapitre 1

Course Examples

Creating a basic java socket to connect a client with a server and sent a welcoming message.
here is an example on how to handle server side requests

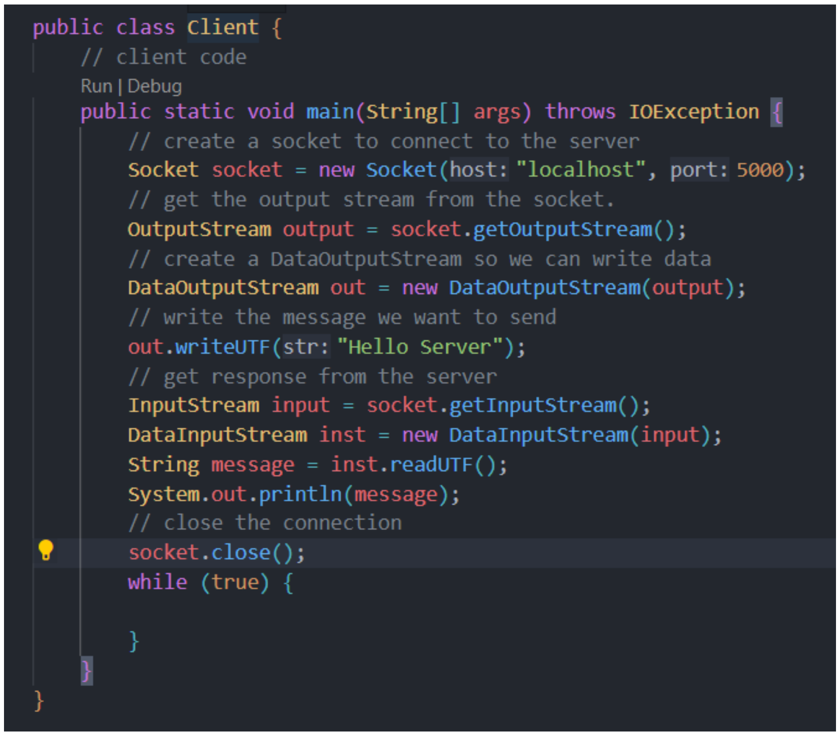


```
public class App {
    Run | Debug
    public static void main(String[] args) throws IOException {
        // start the server
        ServerSocket server = new ServerSocket(port: 5000);
        System.out.println(x: "Server started");
        while (true) {
            try {
                Socket client = server.accept();
                System.out.println(x: "Client connected");
                // get the input stream from the client
                InputStream input = client.getInputStream();
                // create a DataInputStream so we can read data from it.
                DataInputStream inst = new DataInputStream(input);
                // read the message from the client
                String message = inst.readUTF();
                System.out.println(message);
                // send a response to the client
                OutputStream output = client.getOutputStream();
                DataOutputStream outst = new DataOutputStream(output);
                outst.writeUTF(str: "Hello Client");
                // close the connection
                client.close();
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
                break;
            }
        }
        server.close();
        System.out.println(x: "Server stopped");
    }
}
```

Figure 1.1 – Server Socket

For the client side, we use a socket class to connect to the same port where the server is running.

port 5000

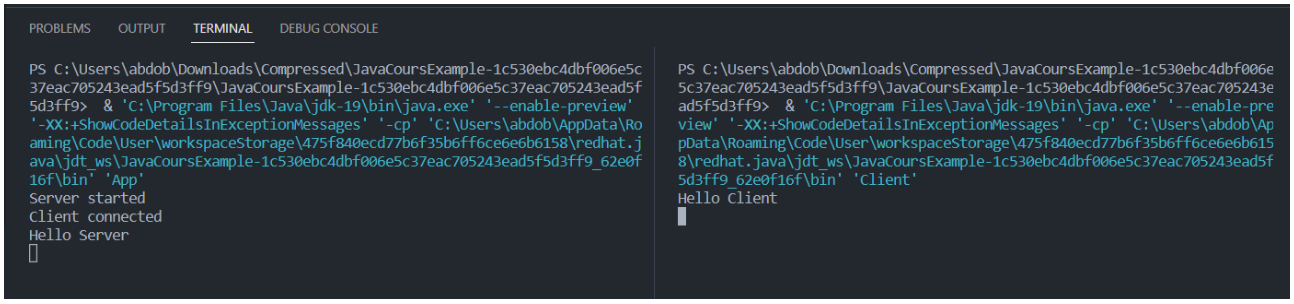


```java
public class Client {
    // client code
    Run | Debug
    public static void main(String[] args) throws IOException {
        // create a socket to connect to the server
        Socket socket = new Socket(host: "localhost", port: 5000);
        // get the output stream from the socket.
        OutputStream output = socket.getOutputStream();
        // create a DataOutputStream so we can write data
        DataOutputStream out = new DataOutputStream(output);
        // write the message we want to send
        out.writeUTF(str: "Hello Server");
        // get response from the server
        InputStream input = socket.getInputStream();
        DataInputStream inst = new DataInputStream(input);
        String message = inst.readUTF();
        System.out.println(message);
        // close the connection
        socket.close();
        while (true) {

        }
    }
}
```

FIGURE 1.2 – Client socket

The above code gives us the following result :



```
PS C:\Users\abdob\Downloads\Compressed\JavaCoursExample-1c530ebc4dbf006e5c
37eac705243ead5f5d3ff9\JavaCoursExample-1c530ebc4dbf006e5c37eac705243ead5f
5d3ff9>  & 'C:\Program Files\Java\jdk-19\bin\java.exe' '--enable-preview'
'-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\abdob\AppData\Ro
aming\Code\User\workspaceStorage\475f840ecd77b6f35b6ff6ce6e6b6158\redhat.j
ava\jdt_ws\JavaCoursExample-1c530ebc4dbf006e5c37eac705243ead5f5d3ff9_62e0f
16f\bin' 'App'
Server started
Client connected
Hello Server
```

```
PS C:\Users\abdob\Downloads\Compressed\JavaCoursExample-1c530ebc4dbf006e
5c37eac705243ead5f5d3ff9\JavaCoursExample-1c530ebc4dbf006e5c37eac705243e
ad5f5d3ff9>  & 'C:\Program Files\Java\jdk-19\bin\java.exe' '--enable-pre
view' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\abdob\Ap
pData\Roaming\Code\User\workspaceStorage\475f840ecd77b6f35b6ff6ce6e6b615
8\redhat.java\jdt_ws\JavaCoursExample-1c530ebc4dbf006e5c37eac705243ead5f
5d3ff9_62e0f16f\bin' 'Client'
Hello Client
```

FIGURE 1.3 – Client Hello World to server

Chapitre 2

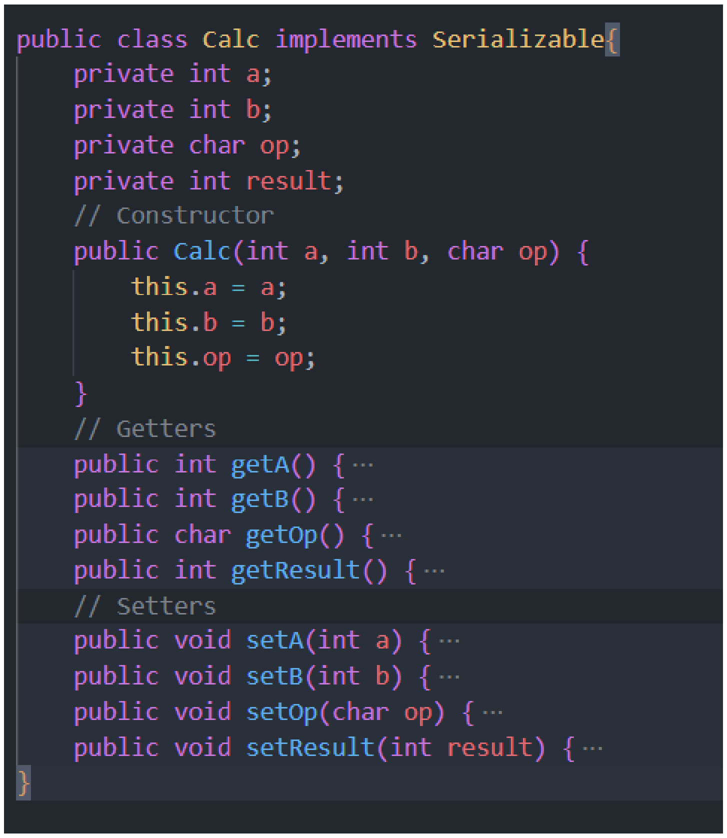
Using Server-Socket to do calculation

2.1 Code explaining

In the first example, we used the java socket only to send a string of text ; now we will use it to send an object.

The idea is to send an object calculation containing two numbers and an operator as a class field along with a field containing the calculation's result.

We will first create the clacul object using the documentation from the collection and Java input-output courses. Wishes that are serializable will allow us to send it objects in both the output and input streams.



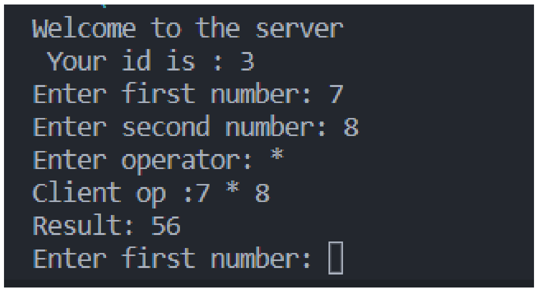
```java
public class Calc implements Serializable{
    private int a;
    private int b;
    private char op;
    private int result;
    // Constructor
    public Calc(int a, int b, char op) {
        this.a = a;
        this.b = b;
        this.op = op;
    }
    // Getters
    public int getA() {···
    public int getB() {···
    public char getOp() {···
    public int getResult() {···
    // Setters
    public void setA(int a) {···
    public void setB(int b) {···
    public void setOp(char op) {···
    public void setResult(int result) {···
}
```

FIGURE 2.1 – Class calculation - serializable

Future code explanations will be attached to the code source.

2.2 results

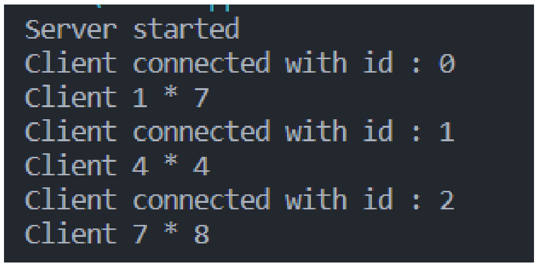
the result will look something like this, On the client side :



```
Welcome to the server
 Your id is : 3
Enter first number: 7
Enter second number: 8
Enter operator: *
Client op :7 * 8
Result: 56
Enter first number: 
```

FIGURE 2.2 – Terminal - client side

and on the server side, the output will look like this :



```
Server started
Client connected with id : 0
Client 1 * 7
Client connected with id : 1
Client 4 * 4
Client connected with id : 2
Client 7 * 8
```

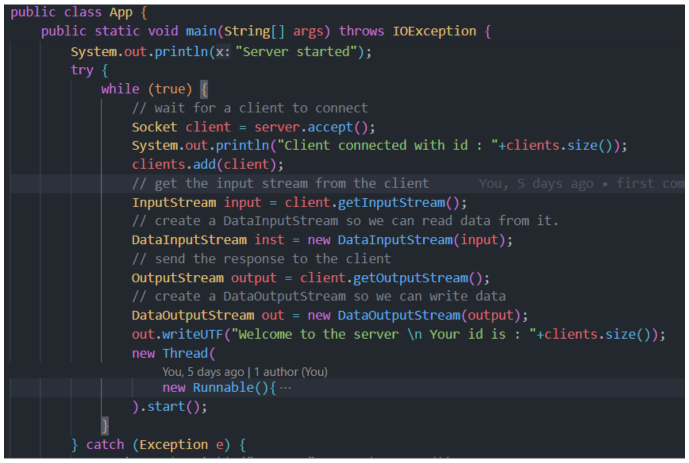
FIGURE 2.3 – Terminal - server side

Chapitre 3

Using Server-Socket in multi-threading

All the previous socket code we have written allows the server to handle only one client, but that is not what we want from a socket server, so we will use multi-threading to allow the server socket to handle multiple clients.

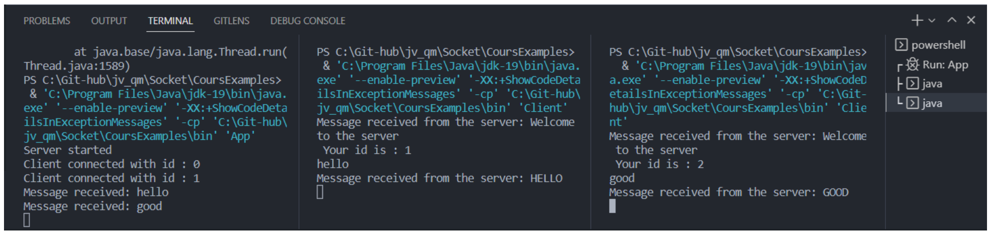
The idea is that once a client connects to the server, we will lunch a thread to handle all the clients requests to the server socket .



```java
public class App {
    public static void main(String[] args) throws IOException {
        System.out.println(x: "Server started");
        try {
            while (true) {
                // wait for a client to connect
                Socket client = server.accept();
                System.out.println("Client connected with id : "+clients.size());
                clients.add(client);
                // get the input stream from the client      You, 5 days ago • first com
                InputStream input = client.getInputStream();
                // create a DataInputStream so we can read data from it.
                DataInputStream inst = new DataInputStream(input);
                // send the response to the client
                OutputStream output = client.getOutputStream();
                // create a DataOutputStream so we can write data
                DataOutputStream out = new DataOutputStream(output);
                out.writeUTF("Welcome to the server \n Your id is : "+clients.size());
                new Thread(
                    You, 5 days ago | 1 author (You)
                    new Runnable(){···
                ).start();
            }
        } catch (Exception e) {
```

FIGURE 3.1 – Multi-Threading serer socket

now let's run this server with multiple clients at the same time and see our result



```
        at java.base/java.lang.Thread.run(
Thread.java:1589)
PS C:\Git-hub\jv_qm\Socket\CoursExamples>
 & 'C:\Program Files\Java\jdk-19\bin\java.
exe' '--enable-preview' '-XX:+ShowCodeDeta
ilsInExceptionMessages' '-cp' 'C:\Git-hub\
jv_qm\Socket\CoursExamples\bin' 'App'
Server started
Client connected with id : 0
Client connected with id : 1
Message received: hello
Message received: good
```

```
PS C:\Git-hub\jv_qm\Socket\CoursExamples>
 & 'C:\Program Files\Java\jdk-19\bin\java.
exe' '--enable-preview' '-XX:+ShowCodeDeta
ilsInExceptionMessages' '-cp' 'C:\Git-hub\
jv_qm\Socket\CoursExamples\bin' 'Client'
Message received from the server: Welcome
to the server
 Your id is : 1
hello
Message received from the server: HELLO
```

```
PS C:\Git-hub\jv_qm\Socket\CoursExamples>
 & 'C:\Program Files\Java\jdk-19\bin\jav
a.exe' '--enable-preview' '-XX:+ShowCodeD
etailsInExceptionMessages' '-cp' 'C:\Git-
hub\jv_qm\Socket\CoursExamples\bin' 'Clie
nt'
Message received from the server: Welcome
 to the server
 Your id is : 2
good
Message received from the server: GOOD
```

FIGURE 3.2 – Testing multi-threading

Chapitre 4

Chatting Java-FX Application

4.1 Introduction

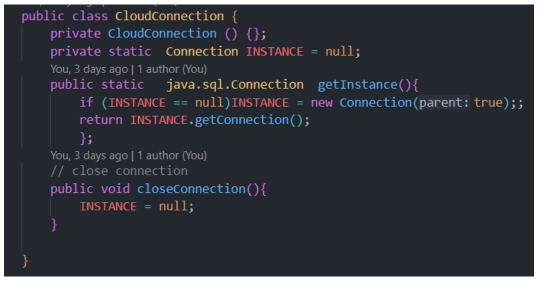
the project is a chat application that uses socket programming to send messages between users and in the main time saves the data to a database.

4.2 Data Base

4.2.1 Design Pattern : Singleton

We have already covered the methods for connecting to a cloud wallet in a previous report, so this time we will focus on the singleton design pattern. In this project, we are utilizing OJDBC with the cloud wallet to connect.

the basic implementation for the Singleton pattern is as shown below :



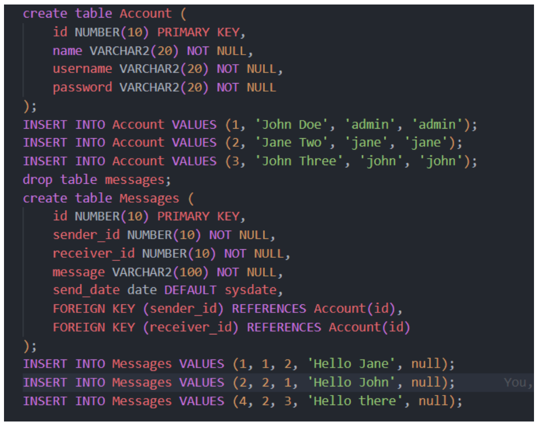
```
public class CloudConnection {
    private CloudConnection () {};
    private static  Connection INSTANCE = null;
    You, 3 days ago | 1 author (You)
    public static   java.sql.Connection  getInstance(){
        if (INSTANCE == null)INSTANCE = new Connection(parent: true);;
        return INSTANCE.getConnection();
        };
    You, 3 days ago | 1 author (You)
    // close connection
    public void closeConnection(){
        INSTANCE = null;
    }

}
```

FIGURE 4.1 – Cloud Connection

4.2.2 SQL Data Base creation instructions

Our database will have two tables : account and messages. We will manage client communications in the account table, which has two keys connected to account table :



```
create table Account (
    id NUMBER(10) PRIMARY KEY,
    name VARCHAR2(20) NOT NULL,
    username VARCHAR2(20) NOT NULL,
    password VARCHAR2(20) NOT NULL
);
INSERT INTO Account VALUES (1, 'John Doe', 'admin', 'admin');
INSERT INTO Account VALUES (2, 'Jane Two', 'jane', 'jane');
INSERT INTO Account VALUES (3, 'John Three', 'john', 'john');
drop table messages;
create table Messages (
    id NUMBER(10) PRIMARY KEY,
    sender_id NUMBER(10) NOT NULL,
    receiver_id NUMBER(10) NOT NULL,
    message VARCHAR2(100) NOT NULL,
    send_date date DEFAULT sysdate,
    FOREIGN KEY (sender_id) REFERENCES Account(id),
    FOREIGN KEY (receiver_id) REFERENCES Account(id)
);
INSERT INTO Messages VALUES (1, 1, 2, 'Hello Jane', null);
INSERT INTO Messages VALUES (2, 2, 1, 'Hello John', null);      You,
INSERT INTO Messages VALUES (4, 2, 3, 'Hello there', null);
```

FIGURE 4.2 – Data Base script

4.2.3 User Interface

Log in

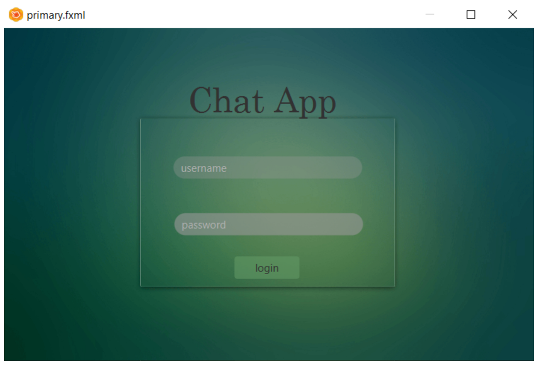
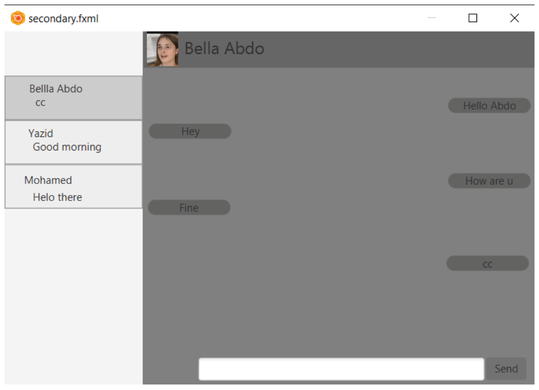


FIGURE 4.3 – Log in interface

Chatting



Bella Abdo

Bellla Abdo
cc

Yazid
Good morning

Mohamed
Helo there

Hello Abdo

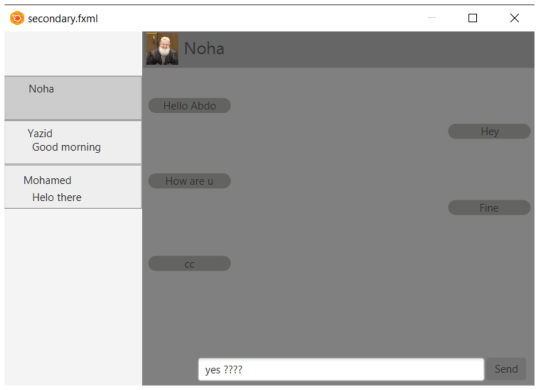
Hey

How are u

Fine

cc

FIGURE 4.4 – Inbox message



Noha

Noha

Yazid
Good morning

Mohamed
Helo there

Hello Abdo

Hey

How are u

Fine

cc

yes ????

FIGURE 4.5 – Replay to message