Progetto Reti di Calcolatori e Laboratorio
“WINSOME: a reWardINg SOcial Media”

Cioni Jacopo
Corso A
581651

Ottobre 2022

Indice

1. Introduzione

Il progetto Winsome è stato sviluppato in Java 17. Per l'implementazione del salvataggio dei dati di persistenza da file è stata utilizzata la libreria Gson (2.8.5) che si occupa della serializzazione e la deserializzazione dei file in Json. La libreria è salvata nel percorso:

winsome/libs/gson-2.8.5.jar

Tutte le classi del progetto Winsome sono state divise in tre moduli principali:

- **module client**, contiene tutte le classi del client.
- **module core,** contiene tutte le classi che il client ed il server hanno in comune.
- **module server**, contiene Tuttle le classi del server.

I metodi main si trovano all'interno dei moduli client e server.

Istruzioni per la compilazione

Per compilare e creare manualmente i file JAR è necessario seguire questo procedimento.
Prima di tutto salvare il contenuto del file .zip in una nuova cartella ed entrare al suo interno. Successivamente dovranno essere eseguiti questi comandi:

- Client:

```
1 mkdir startClient
2 find winsome/client/src/main/java/it/unipi/jcioni/winsome/client winsome/core/src/main/java
  /it/unipi/jcioni/winsome/core -name *.java > forClient.txt
3 javac -d ./startClient/ @forClient.txt
4 cd startClient
5 jar cfe ../WinsomeClient.jar it.unipi.jcioni.winsome.client.Main *
6 cd ..
```

- Server:

```
1 mkdir startServer
2 find winsome/server/src/main/java/it/unipi/jcioni/winsome/server winsome/core/src/main/java
  /it/unipi/jcioni/winsome/core -name *.java > forServer.txt
3 javac -cp "winsome/libs/gson-2.8.5.jar" -d ./startServer/ @forServer.txt
4 cd startServer
5 jar mcfe ../winsome/server/src/main/resources/manifest.txt ../WinsomeServer.jar it.unipi.jc
  ioni.winsome.server.Main *
6 cd ..
```

- Comando per avviare il client:

java -jar WinsomeClient.jar

- Comando per avviare il server:

java -jar WinsomeServer.jar

Per avviare il client/server è possibile eseguire il codice direttamente da riga di comando. Affinché l'esecuzione del programma sia possibile è necessario (soltanto la prima volta) avviare il Server prima del Client. Al primo avvio verrà generata dal Server una cartella chiamata "WinsomeServer" utile al Client per avviarsi. Questa cartella è da considerarsi come uno "storage", infatti al suo interno troviamo tutti i file formato .properties e .json. Al primo avvio il Server si preoccuperà di generare i file .json di sistema ed il proprio file .properties, successivamente il Client, dopo aver individuato "WinsomeServer", realizzerà il proprio file .properties.
I file rimarranno persistenti nel sistema e non sarà necessario crearli nuovamente (a patto che non siano cancellati).

Module Client

La classe principale del client che contiene il metodo main è **Main**. Questa classe si trova nel **module client**, più precisamente all'interno del package:

it.unipi.jcioni.winsome.client

In questo modulo è presente anche un secondo package contenente la parte implementativa del servizio di aggiornamento della lista follower del client.

All'accensione, il Client lavora su due Thread:

- Il Thread socket TCP, utilizzato dal Client per instaurare una connessione TCP con il server
- Il Thread multicast, avviato nel main del Client per gestire le notifiche multicast ricevute in un gruppo le cui coordinate sono definite nel file di configurazione.

Module Server

La classe principale del server che contiene il metodo main è **Main**. Questa classe si trova nel **module server**, più precisamente all'interno del package:

it.unipi.jcioni.winsome.server

In questo modulo è presente anche un secondo package contenente la parte implementativa del servizio di Callback e del servizio di registrazione.

All'accensione, il Server riceve e gestisce le connessioni in ingresso. Per ogni connessione in ingresso viene avviato un Thread che gestisce il pocket aperto e viene inserito in una CachedThreadPool. Escludendo quelli generati per soddisfare le connessioni, vengono creati altri 2 Thread:

- Il Thread rewardsExecutor, avviato prima dell'apertura del socket TCP server nel main ed utilizzato per il calcolo periodico delle ricompense
- Il Thread WinsomeSave, avviato prima dell'apertura del socket TCP server nel main ed utilizzato per il salvataggio periodico dei dati su disco.

2. Connessione TCP - WinsomeClient

Connessione TCP

Per prima cosa, Thread client stabilisce una connessione con il server tramite **Socket** all'indirizzo "localhost". Un socket è un "dispositivo" input/output **bloccante**. Questo strumento fa si che il Thread che lo sta utilizzando sia in grado di bloccarsi sia per le operazioni di lettura che, potenzialmente, quelle di scrittura se il buffer sottostante è pieno. Pertanto, per gestire le richieste lato Server, sarà necessario creare un gruppo di Thread diversi se il server ha un gruppo di socket aperti.

```
// Input stream di bytes
InputStream inputStream = socket.getInputStream();
BufferedReader input = new BufferedReader(new InputStreamReader(inputStream));
// Output stream di bytes
OutputStream outputStream = socket.getOutputStream();
PrintWriter output = new PrintWriter(outputStream,  autoFlush: true);
```

I dati mantengono l'ordinamento FIFO. I metodi che si occupano di ricevere e di inviare dati sono:

```
private static void invia (PrintWriter output, String send) {
    output.println(send);
    output.flush();
}

4 usages  JacopoCioni
private static String ricevi (BufferedReader input) throws IOException {
    String text = input.readLine();
    if (text == null) throw new IOException();
    text = text.replace( oldChar: '$',  newChar: '\n');
    return text;
}
```

WinsomeClient (Main)

In breve, il mainClient si occupa di leggere e validare i parametri dal file .properties e apre un socket TCP con il server. Il mainClient non detiene strutture dati tranne che **followers**, ovvero una HashMap che contiene le informazioni sui followers e si aggiorna tramite il sistema di notifiche basato su RMICallback.
I comandi che compongono il client sono:

- help, fornisce una lista di tutti i comandi utilizzabili dal client
- **register**, permette di registrarsi al servizio tramite RMI
- **login,** permette di collegarsi al servizio
- **logout**, permette di disconnettersi dal servizio
- listfollowers, permette di visualizzare tutti i propri followers. Questo è l'unico metodo gestito direttamente dal Client e presenta controlli sui comandi in ingresso
- **exit**, permette di terminare l'esecuzione del client
- listusers
- listfollowing
- follow
- unfollow
- blog
- post
- showfeed
- showpost
- rewin
- rate
- comment
- wallet
- walletbtc

I comandi che sono stati scritti in grassetto sono quelli che permettono il corretto funzionamento del Client. È stata fornita una breve descrizione per i metodi ritenuti più importanti.

3. Handler

L' Handler è il "cuore" del social Winsome. Ogni istanza di **winsomeHandler** si occupa di gestire una connessione tra un client ed il server. In questa classe troviamo quasi (tutte) le funzionalità del progetto.
La classe si trova nel package:

it.unipi.jcioni.winsome.server

L' Handler vuole come parametro in ingresso al costruttore il Socket del client e winsomeData:

```
public Handler(Socket clientSocket, WinsomeData winsomeData) {
    this.clientSocket = clientSocket;
    this.winsomeData = winsomeData;
    this.session = null;
}
```

WinsomeData è una classe che contiene una **ConcurrentLinkedDeque** di <User> ed ha lo scopo di comportarsi da struttura dati condivisa, in modo tale che ogni istanza di Handler sia in grado di recuperare i riferimenti agli utenti registrati alla piattaforma.
All'attivazione del Thread, l'handler si prepara a ricevere le richieste da parte del client e a gestirle. All'interno dello switch sono presenti tutti i comandi che possono essere gestiti e per ogni **case** è contenuto un codice di controllo della sintassi degli argomenti forniti. In caso di errore vengono utilizzati i metodi **invia** e **ricevi** precedentemente descritti per segnalare al client che la richiesta non è andata a buon fine.

4. Connessione TCP - WinsomeServer

Connessione TCP

La gestione delle richieste ha inizio con l'apertura del **serverSocket.** Nel codice viene generato un loop infinito dove viene richiamato il metodo **.accept()**. Quando viene invocato questo metodo il server si pone in attesa di nuove connessioni e rimane in stato di blocco fino a quando non ne avviene una. Se c'è almeno una richiesta, il processo si sblocca e costruisce un nuovo socket che verrà prima aggiunto alla lista dei sockets in sessione e poi utilizzato come parametro per l'Handler sottomesso alla pool di Thread.

```
while (true) {
    try {
        // bloccante fino a quando non avviene una connessione
        clientSocket = serverSocket.accept();
        sockets.add(clientSocket);
        executor.submit(new Handler(clientSocket, WINSOME_DATA));
    } catch (SocketException e) {
        e.printStackTrace();
        break;
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Per gestire più Thread è stata utilizzata una CachedThreadPool, consigliata quando non è conosciuto a priori il numero di Thread da servire.

WinsomeServer (Main)

In breve, il mainServer si occupa di:

- Leggere e validare i parametri dal file .properties
- Generare (al primo avvio) i file .json per la persistenza dei dati su disco
- Leggere i dati da disco ed inizializzare il social Winsome. Nel social vengono serializzati gli utenti, i posts, i follows ed i followers. In questa fase vengono caricati tutti questi dati nelle rispettive strutture dati che costituiscono il progetto. La prima struttura ad essere formata è WINSOME_DATA (riferimento principale), dopo seguono gli UsersPost, gli UserFollows e gli UserFollowers che verranno, uno ad uno, caricati in WINSOME_DATA

```
// Inizializzazione del social Winsome
// Caricamento di tutti gli Utenti - costruzione di WINSOME_DATA
WINSOME_DATA = new WinsomeData(winsomeUsers);
// Caricamento di tutti i Post degli utenti
WINSOME_DATA.setUsersPosts(winsomePosts);
// Caricamento di tutti i Follows degli utenti
WINSOME_DATA.setUsersFollows(winsomeFollows);
// Caricamento di tutti i Followers degli utenti
WINSOME_DATA.setUsersFollowers(winsomeFollowers);
```

- Avvio del rewardsExecutor per il calcolo delle ricompense
- Inizializzazione RMI/RMICallback
- Avvio Thread di backup
- Apertura del socket e gestione delle richieste

5. RMI

Come è già stato introdotto in precedenza, all'avvio del Client viene creata una connessione con il Server RMI. Per prima cosa il client prende il registro condiviso dal server all'indirizzo "localhost" e poi prende il riferimento dello Stub del server memorizzato sul registro del server.
Successivamente, dopo aver effettuato la login, il client crea una nuova istanza della classe WinsomeNotifyEventImpl e la esporta sullo Stub del client per implementare il meccanismo delle callback.
Lo Stub del client è rappresentato dall'interfaccia WinsomeNotifyEvent.
Lo stub del server è rappresentato dall'interfaccia WinsomeService.

Le classi chiave per l'implementazione di RMI sono:

- WinsomeNotifyEventImpl. Questa classe invia il messaggio di notifica al Client riguardo un nuovo follower e lo aggiunge alla HashMap tenuta localmente dal client. È presente anche il servizio di notifica per l'unfollow di uno User (che verrà anche rimosso).

```java
public void addNotifyEvent(String mainUser, String value) throws RemoteException {
    boolean result = false;
    // Controllo se è già presente
    for (String s: Main.followers.keySet()) {
        if (s.equals(mainUser)) {
            for (String f: Main.followers.get(s)) {
                if (f.equals(value)) {
                    result = true;
                    break;
                }
            }
        }
    }
    if (value != null && !result) {
        Main.followers.get(mainUser).add(value);
        System.out.println("[RMI] - L'utente '"+value+"' ha iniziato a seguirti.");
        return;
    }
    System.out.println("[RMI] - Errore nella ricezione della notifica.");
}
```

- WinsomeServiceImpl. Questa classe contiene il metodo **register**, ovvero una delle funzionalità principali del social che tramite username e password permette di registrare un utente alla piattaforma memorizzando i suoi riferimenti in WINSOME_DATA.
  In questa classe è presente anche il metodo **startFollowers** che permette di inizializzare l'HashMap di followers presente lato client.

- WinsomeCallbackImpl. In questa classe sono presenti i metodi che permettono di registrare o di rimuovere un utente al servizio di notifica per la gestione del follow/follower. Svolgono un ruolo importantissimo i metodi **addUpdate** e **removeUpdate** che permettono di richiamare il metodo che notificherà i vari client ed aggiornerà le loro HashMap di followers.

```java
public static synchronized void addUpdate(String username, String value) throws RemoteException {
    if (clients.containsKey(username)) {
        clients.get(username).addNotifyEvent(username, value);
    }
}

1 usage  JacopoCioni
public static synchronized void removeUpdate(String username, String value) throws RemoteException {
    if (clients.containsKey(username)) {
        clients.get(username).removeNotifyEvent(username, value);
    }
}
```

6. Module

Come detto in precedenza, il progetto si divide in 3 moduli:

- Il modulo **client.** In questa sezione del progetto si trovano tutte le classi relative al funzionamento dei servizi richiesti dal Client. È da considerarsi molto importante la classe WinsomeWalletUpdate che permette al client di aggiungersi ad un gruppo multicast per ricevere le notifiche relative allo Wallet.
- Il modulo **core**. In questa sezione del progetto si trovano tutto le classi che fanno da scheletro alla piattaforma. All'interno del package model possiamo trovare tutte le strutture base della piattaforma. Nella maggior parte dei casi sono state utilizzate strutture dati concorrenti per ovviare al problema della sincronizzazione. È da considerarsi molto importante la classe WinsomeData, elemento che fa da struttura dati dell'intero sistema.
- Il modulo **server.** In questa sezione del progetto si trovano tutte le classi che si occupano di soddisfare le richieste che arrivano dai vari clients. La classe Handler si occupa di eseguire tutte le richieste e la classe WinsomeRewards si occupa di calcolare le ricompense periodiche divise per curatori ed autori. È da considerarsi molto importante la classe WinsomeSave che si occupa di effettuare un salvataggio periodico (ogni minuto) su disco. Il costruttore di questa ultima classe riceve WINSOME_DATA ed esegue il metodo start su se stesso, generando così un Thread del tutto autonomo.