Reti e Laboratorio III
Modulo Laboratorio III
AA. 2023-2024

docente: Laura Ricci

laura.ricci@unipi.it

Correzione Assignment 7

”Dungeon Adventures”

16/11/2023

ASSIGNMENT 7: DUNGEON ADVENTURES

- sviluppare un'applicazione client server in cui il server gestisce le partite giocate in un semplice gioco, “Dungeon adventures” basato su una semplice interfaccia Testuale

- ad ogni giocatore viene assegnato, ad inizio del gioco, un livello X di salute e una quantità Y di una pozione, X e Y generati casualmente

- ogni giocatore combatte con un mostro diverso. Anche al mostro assegnato a un giocatore viene associato, all'inizio del gioco un livello Z di salute generato casualmente

ASSIGNMENT 7: DUNGEON ADVENTURES

il gioco si svolge in round, ad ogni round un giocatore può:

- **combattere con il mostro:** il combattimento si conclude decrementando il livello di salute del mostro e del giocatore. Se LG è il livello di salute attuale del giocatore e MG quello del mostro, tale livello viene decrementato di un valore casuale X, con $0 \leq X \leq LG$. Analogamente, per il mostro si genera un valore casuale K, con $0 \leq K \leq MG$.
- **bere una parte della pozione:** la salute del giocatore viene incrementata di un valore proporzionale alla quantità di pozione bevuta, che è un valore generato casualmente
- **uscire dal gioco:** In questo caso la partita viene considerata persa per il giocatore

il combattimento si conclude quando il giocatore o il mostro o entrambi hanno un valore di salute pari a 0.

se il giocatore ha vinto o pareggiato, può chiedere di giocare nuovamente, se invece ha perso deve uscire dal gioco.

ASSIGNMENT 7: DUNGEON ADVENTURES

sviluppare una applicazione client server che implementi Dungeon adventures

- il server riceve richieste di gioco da parte dei cliente e gestisce ogni connessione in un diverso thread
- ogni thread riceve comandi dal client li esegue. Nel caso del comando “combattere”, simula il comportamento del mostro assegnato al client
- dopo aver eseguito ogni comando ne comunica al client l'esito
- comunica al client l'eventuale terminazione del del gioco, insieme con l'esito
- il client si connette con il server
- chiede iterativamente all'utente il comando da eseguire e lo invia al server. I comandi sono i seguenti 1:combatti, 2: bevi pozione, 3: esci del gioco
- attende un messaggio che segnala l'esito del comando
- nel caso di gioco concluso vittoriosamente, chiede all'utente se intende continuare a giocare e lo comunica al server

ASSIGNMENT 7: IL SERVER

```
/**
 * Reti e Laboratorio III - A.A. 2022/2023
 * Dungeon Adventures
 *
 * Classe Java che rappresenta il server del gioco.
 *
 * Il server gestisce un pool di thread ed esegue un ciclo nel quale:
 * (1) accetta richieste di connessione da parte dei vari client;
 * (2) per ogni richiesta, attiva un thread Worker per interagire con il client;
 *
 */
public class Server {
    public static final String configFile = "server.properties";
    public static int port;
    public static int maxDelay;
    public static final ExecutorService pool = Executors.newCachedThreadPool();
    public static ServerSocket serverSocket;
```

ASSIGNMENT 7: IL SERVER

```
public static void main(String[] args) throws Exception {
    try {
        readConfig();
        serverSocket = new ServerSocket(port);
        Runtime.getRuntime().addShutdownHook(
            new TerminationHandler(maxDelay, pool, serverSocket)
        );
        System.out.printf("[SERVER] In ascolto sulla porta: %d\n", port);
        while (true) {
            Socket socket = null;
            // Accetto le richieste provenienti dai client.
            try {socket = serverSocket.accept();}
            catch (SocketException e) {break;}
            pool.execute(new Worker(socket));
        }
    } catch (Exception e) {
        System.err.printf("[SERVER]: %s\n",e.getMessage());
        System.exit(1);
    }
}
```

ASSIGNMENT 7: CONFIGURARE IL SERVER

```java
/**
 * Metodo che legge il file di configurazione del server.
 * @throws FileNotFoundException se il file non esiste
 * @throws IOException se si verifica un errore durante la lettura
 */

public static void readConfig() throws FileNotFoundException, IOException {
    InputStream input = Server.class.getResourceAsStream(configFile);
    Properties prop = new Properties();
    prop.load(input);
    port = Integer.parseInt(prop.getProperty("port"));
    maxDelay = Integer.parseInt(prop.getProperty("maxDelay"));
    input.close();
}
```

ASSIGNMENT 7: CONFIGURARE IL SERVER

```
#
#File di configurazione del server
#


# Porta di ascolto del server.
port=12000
# Tempo di attesa prima della chiusura del pool di thread.
maxDelay=60000
```

ASSIGNMENT 7: IL WORKER

```
/**
*
 * Il thread Worker si occupa di interagire con un utente durante le partite.
 *
 * Durante una partita, il thread Worker:
 * (1) riceve un comando dall'utente;
 * (2) esegue l'azione richiesta;
 * (3) comunica al client l'esito dell'operazione e lo stato corrente del gioco.
 *
 * I messaggi di risposta inviati dal Worker sono formati da una singola riga
 * con il seguente formato: [stato],[contenuto]\n
 *
 * dove [stato] indica il nuovo stato del gioco dopo l'azione richiesta
dall'utente
 * e [contenuto] include il testo del messaggio di risposta per il client.
 *
 */
```

ASSIGNMENT 7: STATI DEL WORKER

```
/**
 *Reti e laboratorio III - A.A. 2022/2023
 *Dungeon Adventures
 *
 *Tipo enumerato che descrive lo stato di una partita.
 *
 */
public enum Status {
    PLAYING,
    INTERRUPTED,
    WIN,
    LOSE,
    DRAW
}
```

ASSIGNMENT 7: IL WORKER

```
public class Worker implements Runnable {
    private final int initialHealth = 10000;
    private final int initialPotion = 2000;
    private int playerHealth = initialHealth;
    private int enemyHealth = initialHealth;
    private int potion = initialPotion;
    private Status status = Status.PLAYING;
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;

    public Worker(Socket socket) {
        this.socket = socket;
    }
```

ASSIGNMENT 7: IL WORKER

```
public void run() {
    try {
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);
        while (true) {
            game();
            // Se l'utente ha perso o interrotto la partita, termino.
            if (status == Status.LOSE || status == Status.INTERRUPTED) break;
            // Altrimenti, se ha vinto attendo la sua decisione
            String line = in.readLine().toLowerCase();
            if (line == null || !line.equals("y")) break;
            // Resetto le variabili per iniziare una nuova partita.
            reset();
        }
        // Chiudo gli stream e la socket.
        in.close(); out.close(); socket.close();
    } catch (Exception e) {
        System.err.printf("[WORKER] Errore: %s\n", e.getMessage()); }
}
```

ASSIGNMENT 7: GIOCHIAMO

```
public void game() throws IOException {
    System.out.println("[WORKER] Partita iniziata.");
    while (status == Status.PLAYING) {
        String line = in.readLine();
        line = line.toLowerCase();
```

ASSIGNMENT 7: GIOCHIAMO

```
switch (line) {
    case "fight":
        fight();
        break;
    case "drink":
        drink();
        break;
    case "potion":
        remainingPotion();
        break;
    case "leave":
        status = Status.INTERRUPTED;
        String message = "Hai perso!";
        out.printf("%s,%s\n", status.name(), message);
        break;
   default:
        out.printf("%s,Errore: comando non valido.\n", status.name());
        break;
}
```

ASSIGNMENT 7: GIOCHIAMO

```
public void fight() {
    int playerDamage = ThreadLocalRandom.current().nextInt(0, playerHealth+1);
    int enemyDamage = ThreadLocalRandom.current().nextInt(0, enemyHealth+1);
    playerHealth -= playerDamage;
    enemyHealth -= enemyDamage;
    String message = null;

    if (playerHealth > 0 && enemyHealth == 0) {
        status = Status.WIN;
        message = "Hai vinto! :-)";
        out.printf("%s,%s\n", status.name(), message);
        return;
    }
    if (playerHealth == 0 && enemyHealth > 0) {
        status = Status.LOSE;
        message = "Hai perso! :-(";
        out.printf("%s,%s\n", status.name(), message);
        return;
    }
```

ASSIGNMENT 7: GIOCHIAMO

```
    if (playerHealth == 0 && enemyHealth == 0) {
        status = Status.DRAW;
        message = "Pareggio! :-|";
        out.printf("%s,%s\n", status.name(), message);
        return;
    }
    out.printf(
    "%s,Giocatore: %d\tNemico: %d\n", status.name(), playerHealth, enemyHealth);
}
```

ASSIGNMENT 7: GIOCHIAMO

```
public void drink() {
    int quantity = ThreadLocalRandom.current().nextInt(0, potion+1);
    potion -= quantity;
    playerHealth += quantity;
    out.printf(
    "%s,Giocatore: %d\tNemico: %d\n", status.name(), playerHealth, enemyHealth);
}

public void remainingPotion() {
    out.printf("%s,Pozione rimanente: %d\n", status.name(), potion);
}

public void reset() {
    playerHealth = enemyHealth = initialHealth;
    potion = initialPotion;
    status = Status.PLAYING;
}
```

ASSIGNMENT 7: IL CLIENT

```
/*
 * Classe Java che rappresenta il client del gioco.
 *
 * Il client esegue un ciclo nel quale:
 * (1) legge l'input dell'utente da tastiera;
 * (2) invia il messaggio letto al server;
 * (3) riceve (e interpreta) la risposta del server.
 *
 * I comandi supportati dal client sono:
 * (1) fight: combatte contro il mostro;
 * (2) drink: beve una certa quantita' di pozione;
 * (3) potion: visualizza la quantita' di pozione rimanente;
 * (4) leave: abbandona la partita corrente.
 */
```

ASSIGNMENT 7: IL CLIENT

```
public class Client {
    // Percorso del file di configurazione del client.
    public static final String configFile = "client.properties";
    // Variabile globale che rappresenta lo stato corrente.
    public static Status status = Status.PLAYING;
    // Nome host e porta del server.
    public static String hostname; // localhost
    public static int port; // 12000
    // Socket e relativi stream di input/output.
    private static Scanner scanner = new Scanner(System.in);
    private static Socket socket;
    private static BufferedReader in;
    private static PrintWriter out;
```

ASSIGNMENT 7: IL CLIENT

```
public static void main(String[] args) {
    try {
        readConfig();
        socket = new Socket(hostname, port);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);
        while (true) {
            game();
            // Se la partita e' terminata con una sconfitta oppure
            // e' stata interrotta volontariamente dall'utente, esco dal ciclo.
            if (status == Status.LOSE || status == Status.INTERRUPTED) break;
            // Altrimenti chiedo se si vuole effettuare una nuova partita
            // e invio la risposta al server.
            System.out.printf("Nuova partita [y/n]?\n> ");
            String command = scanner.nextLine();
            out.println(command);
```

ASSIGNMENT 7: IL CLIENT

```
            if (!command.toLowerCase().equals("y")) break;
                status = Status.PLAYING;
            }
        }
        catch (Exception e) {
            System.err.printf("Errore: %s\n", e.getMessage());
            System.exit(1);
        }
    }
```

ASSIGNMENT 7: LA MOSSA DEL CLIENT

```
public static void game() throws IOException {
    System.out.println("Partita iniziata. Inserisci un comando.");
    while (status == Status.PLAYING) {
        System.out.printf("> ");
        String command = scanner.nextLine();
        out.println(command);
        String reply = in.readLine();
        String[] parts = reply.split(",");
```

ASSIGNMENT 7: LA MOSSA DEL CLIENT

```
            switch (parts[0]) {
                case "WIN":
                    status = Status.WIN;
                    break;
                case "LOSE":
                    status = Status.LOSE;
                    break;
                case "DRAW":
                    status = Status.DRAW;
                    break;
                case "INTERRUPTED":
                    status = Status.INTERRUPTED;
                    break;
                default:
                    break;
            }
            System.out.println(parts[1]);
        }
        System.out.println("Partita terminata.");
    }
```

ASSIGNMENT 7: TERMINAZIONE SERVER

```
/*
 *Classe che implementa l'handler di terminazione per il server.
 *Questo thread viene avviato al momento della pressione dei tasti CTRL+C.
 *Lo scopo e' quello di far terminare il main del server bloccato sulla accept()
 *in attesa di nuove connessioni e chiudere il pool di thread.
 */
public class TerminationHandler extends Thread {
    private int maxDelay;
    private ExecutorService pool;
    private ServerSocket serverSocket;

    public TerminationHandler(
        int maxDelay, ExecutorService pool, ServerSocket serverSocket)
    {
        this.maxDelay = maxDelay;
        this.pool = pool;
        this.serverSocket = serverSocket;
    }
```

ASSIGNMENT 7: TERMINAZIONE SERVER

```
public void run() {
    // Avvio la procedura di terminazione del server.
    System.out.println("[SERVER] Avvio terminazione...");
    // Chiudo la ServerSocket in modo tale da non accettare piu' nuove richieste.
    try {serverSocket.close();}
    catch (IOException e) {
    System.err.printf("[SERVER] Errore: %s\n", e.getMessage());
    }
    // Faccio terminare il pool di thread.
    pool.shutdown();
    try {
        if (!pool.awaitTermination(maxDelay, TimeUnit.MILLISECONDS))
        pool.shutdownNow();
    }
    catch (InterruptedException e) {pool.shutdownNow();}
        System.out.println("[SERVER] Terminato.");
}
```