INDICE: 1 TEXTFIELD LISTENER, 2 LOGGING, 3 LOGGING CON SPECIFICHE, 4 LETTURA DI FILE DI CARATTERI, 5 SCRITTURA DI FILE DI CARATTERI,

6 CLIENT-SERVER, 7 THREAD, 8 THREAD, 9 JOIN, 10 SLEEP INTERRUPT, 11 ACCESSO AI DATI CONDIVISI, 12 CODA LIMITATA CONDIVISA E WAIT,

13 SERVER MULTICLIENT

//1 TEXTFIELD LISTENER

//EsempioTextFieldListener.java

//ascoltatore che ascolta una textfield e all'action event risponde con un messagedialog che mostra cosa è scritto in quel textfield

public class EsempioTextFieldListener implements ActionListener{

private EsempioTextField frame;

//è la frame in cui è contenuto il text field da "ascoltare"

EsempioTextFieldListener(EsempioTextField frame){

this.frame = frame;

}

public void actionPerformed(ActionEvent e){

JOptionPane.showMessageDialog(null, frame.getTextField.getText());

//deve passare per getTextField

//invece di accedere direttamente

}

}

//2 LOGGING

//Loggata.java

import java.util.logging.\*;

public class Loggata{

private static Logger logger = logger.getLogger("etichetta package");

private static void metodoEsempio(){

//questo metodo nella realtà può essere complesso e causare realmente eccezioni (vedi sotto)

System.out.println("..sto eseguendo operazione..");

}

public static void main(String args[]){

try {

Loggata.metodoEsempio();

}catch{

logger.log(Level.WARNING, "problemi nel metodo", ex);

//questo registrerebbe l'eventuale eccezione

}

logger.log(Level.INFO, "fatto");

}

}

//3 LOGGING CON SPECIFICHE

//MainApp.java

import java.util.logging.Logger;

public class MainApp{

//definizione della variabile che fa da logger per la classe. nota che il logger non si crea con new ma si prende con get

final static Logger logger = Logger.getLogger("MainApp");

public void usaLogging(){

logger.log(Level.INFO, "prova info");

logger.log(Level.WARNING, "avvertenza");

logger.log(Level.FINE, "messaggio di debug");

}

public static void main (String[] args){

MainApp ma = new MainApp();

ma.usaLogging();

}

}

//per modificare le specifiche di logging creiamo un file logging.properties nel pacchetto in cui siamo. in questo file incolliamo tutto il contenuto del file logging.properties globale. nelle proprietà del progetto va aggiunta la specifica per indicare di voler usare il nuovo file invece di quello globale: tasto dx sul progetto corrente, proprietà, in Arguments incolla la specifica -Djava.util.logging.config.file=logging.properties per stampare anche il messaggio di livello FINE modifichiamo il nostro logging.file: nella riga java.util.logging.ConsoleHandler.level mettiamo =Fine . tutto questo non funziona da netbeans ma funziona da terminale. ci sono altre specifiche da modificare per salvare tutti i messaggi di logging su un file invece di farli solo apparire sulla console

//4 LETTURA DI FILE DI CARATTERI

//si ha un file, ogni riga del file rappresenta un prodotto per ogni prodotto si ha nome, quantità, prezzo unitario separati da spazi. si crea una classe per esprimere ogni record del file

//InventoryItem.java

import java.util.StringTokenizer;

//questa classe permette di prendere una stringa a separarla token per token usando lo spazio come elemento separatore

public class InventoryItem{

private String prodotto;

private int quantita;

private float prezzo;

public InventoryItem(String prodotto, int quantita, float pr){

this.prodotto = prodotto;

this.quantita = quantita;

this.prezzo = prezzo;

}

//altro costruttore che prende in input riga del file e si occupa della tokenizzazione

public InventoryItem(String linea){

StringTokenizer st = new StringTokenizer(linea);

this.prodotto = st.nextToken();

this.quantita = Integer.parseInt(st.nextToken());

this.prezzo = Float.parseFloat(st.nextToken());

}

@Override

public String toString(){

return "Prodotto: "name+"; Quantità: "+units+"; Prezzo: "+price;

}

//si possono aggiungere i metodi set e get

}

//SOLUZIONE 1: BASATA SU SCANNER

//Principale.java

import java.io.File;

import java.util.Scanner;

public class Principale{

public static void main (String[] args){

//questa soluzione è svolta nel main del file MainApp della soluzione precedente, commentando le due righe scritte in quella soluzione

LinkedList<InventoryItem> ll = new LinkedList<>();

try {

Scanner sc = new Scanner(new File("inventory.dat"));

//questo file va creato o deve esistere

while (sc.hasNext()){

String nuovaLinea = sc.nextLine();

//lettura della nextLine del file

InventoryItem ii = new InventoryItem(nuovaLinea);

//creazione dell'oggetto tokenizzato

logger.log(Level.INFO, ii.toString());

ll.add(ii);

}

sc.close();

}catch(FileNotFoundException e){

logger.log(Level.SEVERE, "File non trovato", e.getMessage());

//usa il file MainApp della lezione precedente

}

}

}

//SOLUZIONE 2: BASATA SU STREAM

//Principale.java

import java.io.File;

import java.util.Scanner;

public class Principale{

public static void main (String[] args){

//questa soluzione è svolta nel main del file MainApp della soluzione precedente, commentando le due righe scritte in quella soluzione

LinkedList<InventoryItem> ll = new LinkedList<>();

try{

BufferedReader br = new BufferedReader(new FileReader(new File("inventory.dat")));

String nuovariga = br.readLine();

while(nuovariga!=null){

InventoryItem ii = new InventoryItem(nuovaRiga);

//creazione dell'oggetto tokenizzato

logger.log(Level.INFO, ii.toString());

ll.add(ii);

nuovaRiga = br.readLine();

}

}catch(FileNotFoundException e){

logger.log(Level.SEVERE, "File non trovato", e.getMessage());

//usa il file MainApp della lezione precedente

}catch(IOException e){

logger.log(Level.SEVERE, "Errore di lettura", e.getMessage());

}

}

}

//5 SCRITTURA DI FILE DI CARATTERI

//si crea una classe per esprimere ogni record del file

//InventoryItem.java

import java.util.StringTokenizer;

//questa classe permette di prendere una stringa a separarla token per token usando lo spazio come elemento separatore

public class InventoryItem{

private String prodotto;

private int quantita;

private float prezzo;

public InventoryItem(String prodotto, int quantita, float pr){

this.prodotto = prodotto;

this.quantita = quantita;

this.prezzo = prezzo;

}

//altro costruttore che prende in input riga del file e si occupa della tokenizzazione

public InventoryItem(String linea){

StringTokenizer st = new StringTokenizer(linea);

this.prodotto = st.nextToken();

this.quantita = Integer.parseInt(st.nextToken());

this.prezzo = Float.parseFloat(st.nextToken());

}

@Override

public String toString(){

return "Prodotto: "name+"; Quantità: "+units+"; Prezzo: "+price;

}

//si possono aggiungere i metodi set e get

}

//Principale.java

import java.io.File;

import java.io.FileWriter;

import java.io.PrintWriter;

import java.util.Scanner;

public class Principale{

public static void main (String[] args){

//questa soluzione è svolta nel main del file MainApp della soluzione precedente, commentando le due righe scritte in quella soluzione

try{

PrintWriter pw = new PrintWriter(new FileWriter(new File("tabelline.txt")));

//questo file che ancora non esiste viene creato

for (int i=1; i<=9; i++){

for (int j=1; j<=10; j++){

pw.print(i\*j + "\t");

}

pw.println();

}

pw.close();

//close fa la flush in automatico

}catch(IOException ex){

Logger.getLogger(MainApp.class.getName().log(Level.SEVERE, "Errore di lettura", ex));

//questo comando non usa il logger creato per la classe ma sfrutta la classe per farlo in automatico

}

}

}

//6 CLIENT-SERVER

//server accetta connessioni su porta 1337. server legge cio che gli viene scritto una riga alla volta e ripete nella stessa connessione cio che ha ricevuto. se riceve una riga quit chiude la connessione e termina esecuzione. il server accetta una sola connessione in ingresso

//nuovo progetto: ComunicazioneRete

//server e client di solito non sono nello stesso progetto ma sono applicazioni distinte

//nuova classe: ServerApp (con main)

//nuova classe: ClientApp (con main)

//ServerApp.java

import java.net.\*;

public class ServerApp{

public static void main(String[] args){

try{

ServerSocket ss = new ServerSocket(8080);

//System.out.println("in attesa di connessione..");

Socket listeningSocket = ss.accept();

//System.out.println("connesso");

Scanner scanner = new Scanner(listeningSocket.getInputStream());

//scanner per leggere sulla connessione

while (true){

String nuovaRiga = Scanner.nextLine();

if (nuovaRiga.equals("quit")) { break; }

System.out.println(nuovaRiga);

}

//alla fine della lettura chiudo il server

listeningSocket.close();

}catch(IOException ex){

Logger.getLogger(ServerApp.class.getName()).log(Level.SEVERE, null, ex));

}

}

}

//ClientApp.java

import java.net.\*;

public class ClientApp{

public static void main(String[] args){

try{

Socket socket = new Socket("127.0.0.1", 8080);

Scanner scannerConsole = new Scanner(System.in);

//scanner per input da console

PrintWriter pw = new PrintWriter(socket.getOutputStream());

//printwriter per stampare sulla connessione

while(scannerConsole.hasNext()){

String nuovaLinea = scannerConsole.nextLine();

pw.println(nuovaLinea);

pw.flush();

//RICORDATI LA FLUSH

if (nuovaLinea.equals("quit")) { break; }

}

//forse ci va anche socket.close();

}catch(IOException ex){

Logger.getLogger(ServerApp.class.getName()).log(Level.SEVERE, null, ex));

}

}

}

//7 THREAD

import java.lang.Runnable;

public class MioRunnable implements Runnable{

private String nome;

public MioRunnable(String n) {

nome = n;

}

public void run(){

//queste istruzioni verranno eseguuite

//quando viene invocata start

for (int i=0; i<10; i++){

System.out.println(nome + ": " + i);

}

System.out.println(nome + ": DONE!");

}

}

import java.lang.Runnable;

import java.lang.Thread;

public class Main{

static void main(String[] args){

MioRunnable ja = new MioRunnable("Jamaica");

//creo l'oggetto Runnable

Thread mioThread = new Thread(ja);

//creo l'oggetto Thread

//il costruttore dell'oggetto Thread prende

//come arg un Runnable

mioThread.start();

//chiamando start sul thread viene invocato run

//da qui in poi ci sono due flussi

}

}

// 8 THREAD

//realizzare classe Runnable che rappresenta un eseguibile che stampa senza andare a capo su output 100 volte un carattere che gli viene passato come parametro. scrivere un main in cui vengono creati e avviati due thread, uno che stampa 0, l'altro che stampa 1

//nuovo progetto "EsempioThread"; nuova classe "PrinterThread"; nuova classe "Main"

//PrinterThread.java

import java.lang.Runnable;

public class PrinterThread implements Runnable{

//CAMPI

private String c;

//COSTRUTTORE

public PrinterThread(String c){

this.c = c;

}

//METODO RUN

@Override

public void run(){

for (int i=0; i<100; i++){

System.out.print(c);

//prova sleep da qui

try{

Thread.sleep((long) (Math.random()\*10));

//math.random restituisce un numero tra 0.0 e 0.1

}catch(InterruptedException ex){

Logger.getLogger(PrinterThread.class.getName()).log(Level.SEVERE, null, ex);

}

//a qui

}

}

}

//Main.java

import java.lang.Runnable;

import java.lang.Thread;

public class Main{

public static void main(String[] args){

//CREO DUE RUNNABLE

PrinterThread p0 = new PrinterThread("0");

PrinterThread p1 = new PrinterThread("1");

//CREO DUE THREAD A CUI PASSO UN RUNNABLE NEL COSTRUTTORE

//FACCIO PARTIRE I THREAD CON START

//START INVOCA RUN DELL'OGGETTO RUNNABLE

Thread t0 = new Thread(p0);

t0.start();

Thread t1 = new Thread(p1);

t1.start();

}

}

// 9 JOIN

//Countdown.java

package countdown;

public class Countdown implements Runnable{

private String id;

private int countdown;

public Countdown(String nome, int partenzaCountdown){

id = nome;

countdown = partenzaCountdown;

}

private String status(){

//ONE LINE IF STATEMENT

return id + ": "+(countdown<0 ? countdown : "Go!") + "\n";

}

public void run(){

while (countdown >= 0){

System.out.print(status());

countdown --;

try{

Thread.sleep(100);

}catch(InterruptedException e){

}

}

}

}

//MainConcorrenteConJoin.java

package countdown;

public class MainConcorrenteConJoin{

public static void main (String[] args){

Countdown h = new Countdown("Houston", 10);

Countdown c = new Countdown("Cape Canaveral", 10);

Thread t1 = new Thread(h);

Thread t2 = new Thread(c);

t1.start();

t2.start();

try{

//QUESTE DUE JOIN SIGNIFICANO CHE IL THREAD MAIN ASPETTA ENTRAMBI I COUNTDOWN DI T1 E T2 PRIMA DI STAMPARE

t1.join();

t2.join();

}catch(InterruptedException e){

//codice da eseguire se il main viene risvegliato da una interrupt prima di aver fatto join

}

System.out.println("Si parte!")

}

}

//10 SLEEP INTERRUPT

//SleepInterrupt.java

public class SleepInterrupt implements Runnable {

public void run(){

try{

System.out.println("in run()");

Thread.sleep(20000);

//20 secondi

System.out.println("in run() - woke up");

}catch(InterruptedException e){

System.out.println("in run()"- interrupted while sleeping);

return;

}

System.out.println("done");

}

}

//Main.java

public class Main{

public static void main(String[] args){

SleepInterrupt si = new SleepInterrupt();

Thread t = new Thread(si);

t.start();

try {

Thread.sleep(2000);

//2 secondi

}catch(InterruptedException x){

//...

}

System.out.println("in main() - interrupting other thread");

t.interrupt();

System.out.println("in main() - leaving");

}

}

// 11 ACCESSO AI DATI CONDIVISI

//Contatore.java

public class Contatore(){

private int valore;

public synchronized void incrementa(){

int tmp = valore;

try{

Thread.sleep(10);

}catch(InterruptedException e){

//...

}

valore = tmp+1;

System.out.println(Thread.currentThread().getName() +

"il contatore segna: " + valore);

}

public synchronized int getValore(){

return valore;

}

}

//quando un thread accederà ad un oggetto di tipo Contatore, i suoi metodi faranno in modo da proteggere la variabile valore grazie alla segnatura synchronized, che assicura che un solo thread per volta acceda al singolo oggetto

//12 CODA LIMITATA CONDIVISA E WAIT

//BoundedQueue.java

public class BoundedQueue<T>{

private List<T> elements;

private int maxSize;

public BoundedQueue(int capacity){

elements = new ArrayList<T>;

maxSize = capacity;

}

public synchronized T remove() throws InterruptedException {

while (elements.size()==0){ wait(); }

//finchè è vuoto non posso togliere

T rimosso = elements.remove(0);

notifyAll();

return rimosso;

}

public synchronized void add(T nuovo) throws InterruptedException {

while (elements.size()==maxSize){ wait(); }

//finchè è pieno non posso aggiungere

elements.add(nuovo);

notifyAll();

}

}

//Producer.java

public class Producer implements Runnable{

private String greeting;

private BoundedQueue<String> queue;

private int greetingCount;

private static final int DELAY = 10;

public Producer(String aGreeting, BoundedQueue<String> aQueue, int count){

greeting = aGreeting;

queue = aQueue;

greetingCount = count;

}

public void run(){

try{

int i=1;

while (i<= greetingCount){

queue.add(i + ": " + greeting);

i++;

Thread.sleep((int) (Math.random()\*DELAY));

}

}catch(InterruptedException ex){

//...

}

}

}

//Consumer.java

public class Consumer implements Runnable{

private BoundedQueue<String> queue;

private int greetingCount;

private static final int DELAY = 10;

public Consumer(BoundedQueue<String> aQueue, int count){

queue = aQueue;

greetingCount = count;

}

public void run(){

try{

int i=1;

while (i<= greetingCount){

Object greeting = queue.remove();

System.out.println(greeting);

i++;

Thread.sleep((int) (Math.random()\*DELAY));

}

}catch(InterruptedException ex){

//...

}

}

}

//ThreadTester.java

public class ThreadTester{

public static void main(String[] args){

BoundedQueue<String> queue = new BoundedQueue<String>(10);

final int GREETING\_COUNT = 100;

Runnable run1 = new Producer("hello!", queue, GREETING\_COUNT);

Runnable run2 = new Producer("bye!", queue, GREETING\_COUNT);

Runnable run3 = new Consumer(queue, GREETING\_COUNT);

Runnable run4 = new Consumer(queue, GREETING\_COUNT);

Thread t1 = new Thread(r1);

Thread t2 = new Thread(r2);

Thread t3 = new Thread(r3);

Thread t4 = new Thread(r4);

t1.start();

t2.start();

t3.start();

t4.start();

}

}

//13 SERVER MULTICLIENT

// si vuole modificare la classe EchoServer in modo che sia multithread (possa gestire più client contemporaneamente).

si sposta la logica che gestisce la comunicazione con il client in una nuova classe ClientHandler che implementa Runnable (cosi che possa essere passata al thread che gestisce la singola connessione).la classe ClientHandler si occuperà di gestire la comunicazione con il client associato al socket assegnato.

la classe principale del server si occuperà solo di istanziare il ServerSocket, eseguire la accept() e di creare i thread necessari per gestire le connessioni accettate.

//EchoServerClientHandler

import java.net.\*;

public class EchoServerClientHandler implements Runnable{

private Socket socket;

public EchoServerClientHandler(Socket socket){

this.socket = socket;

}

public void run(){

try{

Scanner in = new Scanner(socket.getInputStream());

PrintWriter out = new PrintWriter(socket.getOutputStream);

//leggo e scrivo nella connessione finchè non ricevo quit

while (true){

String line = in.nextLine();

if (line.equals("quit")){

break;

}else{

out.println("Received: "+ line);

out.flush();

}

}

//chiudo gli stream e il socket

in.close();

out.close();

socket.close();

}catch(IOException ex){

System.err.println(ex.getMessage());

}

}

}

//QUESTO SERVER NON è PRESO DALLE SOLUZIONI DEL PROF L'HAI FATTO TU

//EchoServer.java

import java.net.\*;

public class EchoServer{

public static void main(String[] args){

try{

//istanzio la server socket

ServerSocket ss = new ServerSocket(8080);

//mi metto in attesa in un ciclo continuo

while(true){

Socket singleSocket = ss.accept();

//quando l'accept capta una connessione restituisce una socket su cui è connesso un client dall'altra parte istanzio un oggetto della classe che implementa runnable così posso usare il costruttore per passargli la socket attiva e posso metterlo in un thread che ne chiama la run

EchoServerClientHandler singleRunnable = EchoServerClientHandler(singleSocket);

Thread singleThread = new Thread(singleRunnable);

singleThread.start();

}

}catch(IOException ex){

System.err.println(ex.getMessage());

}

}

}