

# PROGETTO DI INFORMATICA

Appena visto il progetto ho notato che molta gente si è spaventata a morte!!! Ho pensato che potesse tornarvi utile una sorta di “documentazione” per capire come gestirvi e organizzarvi per realizzare il progetto senza perdere troppo tempo!

Non posso darvi spezzoni di codice senno finiamo per essere bocciati tutti, ma posso darvi delle dritte su come dividere il codice (che vi assicuro: è la parte più importante di ogni programma!).

Sia chiaro: NON è l'unica soluzione, anzi probabilmente ce n'è di più semplici e belle; è solo un aiuto che regalo per Natale.

## CONSIGLIO PRELIMINARE

Dividere un programma in fasi (che alla fine significa dividerlo in funzioni che rendano più leggibile e più semplice il codice) semplifica di molto il lavoro del programmatore a discapito del “debugging” (quando bestemmate perché non sapete dove è l'errore e iniziate a cercarlo disperatamente tra le righe del vostro codice).

Immaginate di dividere il codice in 5, 10, 15 blocchi di funzioni lunghe 40-50 righe. Immaginatevi ora che trovate qualcosa che non va, qualche risultato sbagliato. Dove cercate? Esatto: bestemmie!

Quello che vi consiglio è quindi di testare BENE ogni funzione, ogni piccolezza con TANTI test. Ad esempio, come vi scriverò sotto, la prima fase è ovviamente ricavare la matrice dal file che creerà il prof e inserirla in una matrice nel programma. Prima di passare alla fase successiva, verificate con dei test che funzioni correttamente: fatevi qualche matrice a mano, provate a modificare i dati e vedere se, stampandola sulla console, dia i dati giusti... Una volta certi che quella parte funziona correttamente, andate avanti. Così saprete che quando trovate degli errori derivano dalle ultime cose che avete scritto, e non da robe precedenti.

Capito? Bene.

## FASE 1 – LEGGERE IL FILE

La prima cosa da fare ovviamente è leggere il file. Ci servono i dati contenuti nel file: il pezzo e la matrice.

Facciamo una funzione “Leggi\_pezzo(String file)” che va a leggere il nome del pezzo contenuto nel file e lo va ad inserire in una variabile stringa; e facciamo una funzione “Leggi\_matrice(String file)” che legga le dimensioni della matrice e che vada a leggere ogni cella della matrice contenuta nel file e la inserisca in una matrice che creiamo noi.

(Neanche a dirlo: i nomi potete cambiarli e siete liberi di non seguire o di cambiare le idee proposte)

L’idea di questa divisione è che la funzione appaia tipo:

```
public static boolean testCessMoves(String fileName){  
    String pezzo = Leggi_pezzo(fileName);  
    int [][] matrice = Leggi_matrice(fileName);  
}
```

Capite bene che il codice è estremamente leggibile e demanda i problemi e gli errori all’interno delle due funzioni.

Non è necessario vi spieghi come si ricavano questi dati, lo abbiamo fatto con il proff ed è sufficiente che voi copiate e incollate il codice (letteralmente).

## FASE 2 – ARRAY DI MOSSE

Questa parte è **fondamentale** per il funzionamento del programma. Si tratta di gestire le mosse con una certa logica per poi capire se possono essere state (o meno) effettuate da uno dei pezzi degli scacchi.

La cosa che mi è sembrata più comoda è quella di tirare fuori le mosse dalla matrice contenuta nel file (che avrete già letto) e farci una matrice 3xn in cui tengo conto del numero di mossa, dell’indice di riga e di colonna. Cioè:

MOSSA	1	...	7
RIGA	41	...	3
COLONNA	13	...	8

Dove la prima mossa è alla riga 41 e alla colonna 13 e l’ultima alla riga 3 e colonna 8.

Qui ovviamente nasce il problema: “Quanto è grande questa matrice?” Risposta: “BHO”. Le righe devono essere 3 ma il numero di colonne dipende dal numero di mosse. Ci sono potenzialmente 2 soluzioni:

1. Prima cercate nella matrice il numero più grande che trovate: quello lì è il numero di mosse e quindi il numero di colonne
2. Fate un array di dimensione 3x1 e mano a mano che trovate una mossa lo aumentate di 1 colonna

La prima soluzione è più comoda, ha solo il difetto di essere lentina, ma cambia poco.

La seconda è la più ottimale perché scandite la matrice solo una volta. È più complicata perché per aumentare di una colonna una matrice dovete:

- Creare una matrice temporanea con uguali dimensioni
- Copiare nella matrice temporanea la roba che c'è nella matrice mosse
- Ricreare la matrice mosse con una dimensione in più:  
`matrice = new int[3][matrice[0].length + 1]`
- Ricopiare i dati della matrice temporanea in quella delle mosse
- Aggiungere nella nuova colonna l'ultima mossa che avete trovato

È un po' sbattosa...

## FASE 2.a – ORDINAMENTO DELLE MOSSE

Ora abbiamo un array con tutte le mosse eseguite, ma non è ordinato! Quando controllate se una mossa è lecita per un determinato pezzo dovete farlo in modo sequenziale, non con mosse a caso!

Questo non obbliga a ordinare l'array in modo da avere le mosse dalla 1 alla N, ma farlo non è una cattiva idea. Per ordinarlo potete inventarvi un algoritmo (CATTIVA IDEA) oppure prenderne di già fatti dal web. Devo ancora chiedere a Zunino quanto sia legittima questa cosa, ma da informatico (si fa per dire) penso di sì perché gli algoritmi di ordinamento degli array sono già noti e già implementati. Voglio dire che se un programmatore in una azienda deve ordinare un array per qualunque ragione è legittimato a copiare e incollare il codice di un ordinamento su internet perché non sono invenzione di aziende, corporazioni o simili, sono risultati matematici/informatici noti e senza copyright. Chiederò a Zunino comunque... Nel caso sceglieste di ordinare l'array vi consiglio di sfruttare un algoritmo chiamato "[Bubble sort](#)", è abbastanza lento ma è il più facile da leggere, e questa è una caratteristica importante perché dovete farci delle modifiche: quando troverete l'implementazione java del bubble sort questa funzionerà per un array 1xn, cioè per un vettore. Dovete stare attenti ad aggiungere che vi scambi anche la 2° e la 3° riga, altrimenti farete un casino incredibile.

```
public void bubbleSort(int [] array) {
    for(int i = 0; i < array.length; i++) {
        boolean flag = false;
        for(int j = 0; j < array.length-1; j++) {
            //Se l' elemento j e maggiore del successivo allora
            //scambiamo i valori
            if(array[j]>array[j+1]) {
                int k = array[j];
                array[j] = array[j+1];
                array[j+1] = k;
                flag=true; //Lo setto a true per indicare che é avvenuto uno scambio
            }
        }
        if(!flag) break; //Se flag=false allora vuol dire che nell' ultima iterazione
            //non ci sono stati scambi, quindi il metodo può terminare
            //poiché l' array risulta ordinato
    }
}
```

Questa è l'implementazione java del bubble sort (**ATTENZIONE:** io non ho implementato questa ma una simile. Non posso assicurarvi che funzioni). L'unico scambio dei valori avviene nella riga evidenziata, ma lì sta cambiando solo i due valori di un vettore e il risultato è tipo:  $(x, y, z) \rightarrow (y, x, z)$ . Se non modifichiamo il codice, il risultato che otteniamo sarà:  $\begin{pmatrix} x & y & z \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{pmatrix} \rightarrow \begin{pmatrix} y & x & z \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{pmatrix}$  e non ci piace.

L'alternativa all'ordinamento dell'array è: quando confrontate le 2 mosse per verificare che siano legittime dovrete cercarle con un ciclo ogni volta. Mi spiego meglio:

2	4	3	1
$i_2$	$i_4$	$i_3$	$i_1$
$j_2$	$j_4$	$j_3$	$j_1$

Se questa è la matrice delle mosse, per verificare che, ad esempio, una torre possa eseguire questi spostamenti sulla scacchiera devo partire dalla mossa 1. Per trovare la mossa 1 devo fare un ciclo che mi trovi la sua posizione nella matrice, da qui so che la relativa posizione sulla scacchiera (cioè  $i_1$  per le righe e  $j_1$  per le colonne) avranno lo stesso indice di colonna. Successivamente devo rifare il ciclo per trovare a quale indice di colonna trovo la seconda mossa. Ora che so dove sono la 1° e la 2° le confronto in base a che pezzo sto valutando. Verificato che la mossa 1 e 2 sono legittime, devo trovare la posizione della mossa 3 e confrontarla con la mossa 2 e via così.

### FASE 3 – ERRORI INIZIALI

Non tutti gli errori necessitano di conoscere il pezzo degli scacchi. Errori di questo tipo sono:

1. Numeri negativi nella matrice
2. Numeri doppi nella matrice
3. Manca la prima mossa
4. Mancano delle mosse

#### 1 Numeri negativi

Il primo errore è facile da verificare: ora che abbiamo le mosse all'interno di un array basta verificare che il numero di mossa sia maggiore di 0 e abbiamo finito. Non ci serve neanche sapere le sue coordinate.

#### 2 Numeri doppi

Di nuovo, basta controllare che nell'array non ci siano doppi numeri nelle mosse ignorando le coordinate

#### 3 Manca la prima mossa

Cercate il numero 1 tra i numeri delle mosse e siete ok

#### 4 Mancano delle mosse

Se le mosse vanno dalla 1 alla 3 senza il 2 è sbagliato

NOTA BENE: Se avete ordinato l'array tutte le 4 situazioni sono veramente stupide perché: i numeri negativi saranno il primo elemento della vostra lista, li trovate subito. Eventuali numeri doppi sono adiacenti. Se manca la prima mossa nell'array c'è un numero diverso da 1 al primo elemento. Se mancano delle mosse: basta verificare che il numero in posizione  $i$  sia uguale al numero in posizione  $i-1$  sommandoci 1.

Questi 4 tipi di errori avranno ognuno una funzione dedicata, in modo da rendere leggibile il codice. Insomma, se non sapete cosa fare, fate una funzione in più.


#### FASE 4 – TEST SULLA LEGITTIMITA' DELLE MOSSE

È arrivato il momento di testare le mosse in base al pezzo. Ovviamente dovrete fare una funzione per ogni pezzo e la logica da seguire è assolutamente matematica (quindi roba che ci piace).

La situazione è:

Ho un array con tutte le mosse e le relative posizioni sulla matrice: come faccio a capire se un pezzo può avere fatto quelle mosse seguendo le regole di movimento?

La cosa è abbastanza facile, basta visualizzare le mosse legittime. Prendo in esempio il re:

	J-1	J	J+1
I-1	A	B	C
I	D		E
I+1	F	G	H

Il re può spostarsi solo nelle caselle adiacenti. Consideriamo che gli indici della posizione della cella in cui parte il re siano  $(i, j)$ . Se si sposta nella casella A i suoi indici di posizione saranno  $(i-1, j-1)$ , nella cella B saranno  $(i-1, j)$  e via così:

A	$i-1$	$j-1$
B	$i-1$	$j$
C	$i-1$	$j+1$
D	$i$	$j-1$
E	$i$	$j+1$
F	$i+1$	$j-1$
G	$i+1$	$j$
H	$i+1$	$j+1$

Un po' a modi fisica andiamo a togliere dalla posizione finale quella iniziale: in A otteniamo

$$(i,j)-(i-1,j-1) = (1,1)$$

Ripetiamo per tutte le celle e otteniamo:

Questa è la variazione delle righe e delle colonne, cioè di quante caselle si è spostato rispetto alle righe e

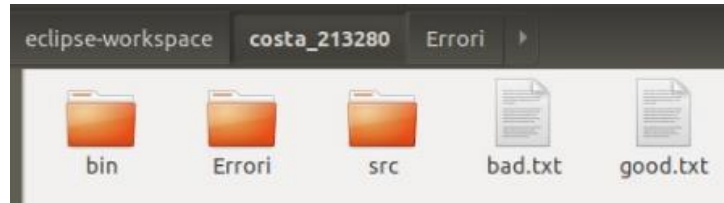
	$\Delta i$	$\Delta j$
A	1	-1
B	1	0
C	1	1
D	0	-1
E	0	1
F	-1	-1
G	-1	0
H	-1	1

rispetto alle colonne. Notiamo che il valore oscilla tra 1 e -1: questo è il criterio.

Cioè, se noi facciamo la differenza tra la mossa successiva e quella precedente in termini di indici di posizione e otteniamo risultati con valori diversi da -1, 0 e 1 allora quella matrice è sbagliata. In termini di valore assoluto: se i delta  $i$  e  $j$  in valore assoluto sono maggiori di 1 la mossa non è valida. Lascio a voi l'implementazione e le regole degli altri pezzi che seguono la stessa logica.

## TEST A RAFFICA

Vi esco anche un piccolo spezzone di codice per fare dei test in massa. Non è niente di particolare, semplicemente un ciclo che ripete il test tante volte. La cosa utile di questo test è che i file che la vostra funzione valuta in modo incorretto vengono salvati in una cartella chiamata "Errori" **CHE DOVETE CREARE VOI** nella stessa directory del vostro progetto:



```
import java.io.*; //DOVETE AGGIUNGERE QUESTA IMPORT ALL'INIZIO DEL VOSTRO PROGRAMMA

public static void copia_file(String file, int num) { //QUESTA FUNZIONE VA MESSA FUORI DAL MAIN
(PERCHÉ È UNA FUNZIONE...)
    InputStream inStream = null;
    OutputStream outStream = null;
    try{
        File afile =new File(file);
        File bfile =new File("./Errori/errore"+num+".txt");//NON SI CREA DA SOLA
        inStream = new FileInputStream(afile);
        outStream = new FileOutputStream(bfile);
        byte[] buffer = new byte[1024];
        int length;
        //copy the file content in bytes
        while ((length = inStream.read(buffer)) > 0){
            outStream.write(buffer, 0, length);
        }
        inStream.close();
        outStream.close();
    }catch(IOException e){
        e.printStackTrace();
    }
}

public static void main(String[] args) {
    //QUESTA PARTE DI CODICE LA INSERITE NEL MAIN (O IN UNA FUNZIONE A PIACIMENTO)
    int successi=0;
    int i;
    for(i=0;i<10000;) { //POTETE CAMBIARE IL NUMERO DI TEST CAMBIANDO QUESTO NUMERINO
        //TEST DI MATRICI CORRETTE
        Helper.generateMoveFile("good.txt");
        if(testChessMoves("good.txt")) {
            successi++;
            System.out.println("SUCCESSO! - Test eseguiti: "+(i+1)+" Passati: "+successi);
        }
        else {
            System.out.println("FALLIMENTO! - Test eseguiti: "+(i+1)+" Passati: "+successi);
            copia_file("good.txt",i);
        }
        i++;
        //TEST DI MATRICI SBAGLIATE
        Helper.generateNonMoveFile("bad.txt");
        if(!testChessMoves("bad.txt")) {
            successi++;
            System.out.println("SUCCESSO! - Test eseguiti: "+(i+1)+" Passati: "+successi);
        }
        else {
            System.out.println("FALLIMENTO! - Test eseguiti: "+(i+1)+" Passati: "+successi);
            copia_file("bad.txt",i);
        }
        i++;
    }
}
```