**Esercizi di programmazione per il capitolo 7**

LA SUA PAGINA CONTIENE diversi esercizi per il Capitolo 7 in [Introduzione alla programmazione con Java](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/index.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhj0yZy6RtTLSTPegGqHMWpC7MF8hw) . Per ogni esercizio viene fornito un collegamento a una possibile soluzione. Ogni soluzione include una discussione su come un programmatore può affrontare il problema e punti interessanti sollevati dal problema o dalla sua soluzione, oltre al codice sorgente completo della soluzione.

**Esercizio 7.1:**

Scrivi una subroutine che crei un ArrayList contenente diversi numeri interi casuali nell'intervallo da 1 fino ad un massimo specificato. Il numero di numeri interi e il valore massimo consentito per gli interi devono essere parametri della subroutine. Scrivi una routine main () per testare la tua subroutine.

[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex1-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhg6hVsHZR9ACIg2MP9Zgn8RJqaPUg)

**Esercizio 7.2:**

Supponiamo che M sia un array bidimensionale con righe R e colonne C. La trasposizione di M è definita come una matrice T che ha colonne C e colonne R tali che T [i] [j] = M [j] [i] per ogni i e j . Scrivi una funzione che accetta un array di tipo int [] [] come parametro e restituisce la trasposizione di quell'array. (Supponiamo che il parametro sia un tipico array 2D in cui tutte le righe abbiano la stessa lunghezza.) Scrivi anche una subroutine per stampare un array 2D di numeri interi in righe e colonne pulite e includere una routine main () per testare il tuo lavoro.

[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex2-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhi1gT9JdevEYdF5tDh59XkwXcheXA)

**Esercizio 7.3:**

Nella [sottosezione 7.4.4](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/s4.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhi13-ZomwvsXhL0bOKyJxn63ppUOQ#arrays.4.4) , si menziona che il metodo di ordinamento standard Arrays.sort () è molto più rapido ed efficiente dell'ordinamento di selezione. Scrivi un programma per testare questo reclamo. Per essere specifici, il tuo programma dovrebbe creare un grande array pieno di numeri reali casuali. Dovrebbe usare sia Arrays.sort () che selectionSort () per ordinare l'array, e dovrebbe calcolare il tempo necessario per eseguire ciascun ordinamento. Inoltre, dovrebbe fare la stessa cosa per una vasta gamma di *stringhe* casuali. Per trovare le ore, è possibile utilizzare System.currentTimeMillis () (vedere [Sottosezione 2.3.1](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c2/s3.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhgsvITmAqn1KQT-QbC35IUmMrFONg#basics.3.1) e l'esempio [*TimedComputation.java*](http://math.hws.edu/javanotes/source/chapter2/TimedComputation.java) ).

[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex3-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhjMRN8WkA8g5S937ldVyBGYTgfuaw)

**Esercizio 7.4:**

Nell'esercizio [6.1](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c6/ex1-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhggeXUarkWAy5Cvy9RBqrDhxzhCCA) , hai scritto un programma SimpleStamperWithDrag che consente all'utente di posizionare rettangoli rossi e ovali blu in un pannello facendo clic e trascinando il mouse. Tuttavia, quel programma non memorizza alcuna informazione su ciò che è stato disegnato, quindi il pannello non può ridisegnare correttamente se stesso. Rivedere il programma per utilizzare un *ArrayList*per memorizzare i dati sul contenuto del pannello. Tutto il disegno dovrebbe essere fatto in un metodo paintComponent () .

[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex4-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhjQ_RARGB94mF2r84c-Wj3fpnmOxw)

**Esercizio 7.5:**

Scrivi un programma che leggerà una sequenza di numeri reali positivi inseriti dall'utente e stamperà gli stessi numeri in ordine decrescente dal più piccolo al più grande. L'utente immetterà uno zero per contrassegnare la fine dell'input. Supponiamo che vengano inseriti al massimo 100 numeri positivi. Non utilizzare alcuna funzione incorporata come Arrays.sort () . Fai l'ordinamento da solo.

[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex5-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhhy6vLhERBbQ1-_z0zXRro_eVdRdQ)

**Esercizio 7.6:**

Il programma di esempio [*RandomArt.java*](http://math.hws.edu/javanotes/source/chapter6/RandomArt.java) della [sottosezione 6.4.1](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c6/s4.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhhAK5w4Wbiha6w3ISh4aagjZfr3yA#GUI1.5.1) mostra un "artwork" casuale differente ogni quattro secondi. Ci sono tre tipi di "arte", una fatta da linee, una da cerchi e una da quadrati pieni. Tuttavia, il programma non salva i dati per l'immagine mostrata sullo schermo. Di conseguenza, l'immagine non può essere ridisegnata quando necessario. Infatti, ogni volta che viene chiamato paintComponent () , viene disegnata una nuova immagine.

Scrivi una nuova versione di RandomArt.java che salva i dati necessari per ridisegnare le sue immagini. Il metodo paintComponent () dovrebbe semplicemente utilizzare i dati per disegnare l'immagine. I nuovi dati dovrebbero essere ricalcolati solo ogni quattro secondi, in risposta a un evento dal timer che guida il programma.

Per rendere questo interessante, scrivi una classe separata per ciascuno dei tre diversi tipi di arte. Scrivi anche una classe astratta per fungere da classe base comune per le tre classi. Poiché tutti e tre i tipi di arte usano uno sfondo grigio casuale, il colore dello sfondo può essere definito nella loro superclasse. La superclasse contiene anche un metodo draw () che disegna l'immagine; questo è un metodo astratto perché la sua implementazione dipende dal particolare tipo di arte che viene disegnata. La classe astratta può essere definita come:

classe astratta privata ArtData {

Colore backgroundColor; // Il colore di sfondo per l'arte.

ArtData () {// Il costruttore imposta il colore di sfondo come grigio casuale.

int x = (int) (256 \* Math.random ());

backgroundColor = new Color (x, x, x);

}

abstract void draw (Grafica g); // Disegna questo disegno.

}

Ognuna delle tre sottoclassi di ArtData deve definire il proprio metodo draw () . Deve anche definire variabili di istanza per contenere i dati necessari per disegnare l'immagine. Suggerisco di creare dati casuali per l'immagine nel costruttore della classe, in modo che la costruzione dell'oggetto crei automaticamente i dati per la grafica casuale. (Un problema con questo è che non è possibile creare i dati finché non si conosce la dimensione del pannello, quindi non è possibile creare un oggetto ArtData nel costruttore del pannello. Una soluzione consiste nel creare un oggetto ArtDataall'inizio del metodo paintComponent () , se l'oggetto non è già stato creato.) In ognuna delle tre sottoclassi, sarà necessario utilizzare uno o più array o ArrayList per memorizzare i dati.

[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex6-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhh1Ub8jZ7d2JLI6lkubaaE-yuPwyw)

**Esercizio 7.7:**

Scrivi un programma che leggerà un file di testo selezionato dall'utente e creerà un elenco alfabetico di tutte le parole in quel file. Tutte le parole devono essere convertite in lettere minuscole e i duplicati devono essere eliminati dall'elenco. L'elenco dovrebbe essere scritto in un file di output selezionato dall'utente. Come discusso nella [sottosezione 2.4.4](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c2/s4.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhhyotgGMROmD1VAcM91EKnUJuaKmg#basics.4.4) , puoi usare *TextIO* per leggere e scrivere file. Utilizzare una variabile di tipo ArrayList <String> per memorizzare le parole. Non è facile separare un file in parole mentre lo leggi. È possibile utilizzare il seguente metodo:

/ \*\*

\* Leggi la parola successiva da TextIO, se ce n'è una. Per prima cosa, salta oltre

\* qualsiasi non-lettere nell'input. Se si incontra una fine del file prima

\* viene trovata una parola, restituisce null. Altrimenti, leggi e restituisci la parola.

\* Una parola è definita come una sequenza di lettere. Inoltre, una parola può includere

\* un apostrofo se l'apostrofo è circondato da lettere su ciascun lato.

\* @return la parola successiva da TextIO, o null se è una fine del file

\* incontrato

\* /

stringa statica privata readNextWord () {

char ch = TextIO.peek (); // Guarda il prossimo carattere in input.

while (ch! = TextIO.EOF &&! Character.isLetter (ch)) {

// Salta oltre le lettere.

TextIO.getAnyChar (); // Leggi il personaggio.

ch = TextIO.peek (); // Guarda il prossimo personaggio.

}

if (ch == TextIO.EOF) // Encountered end-of-file

return null;

// A questo punto, sappiamo che il prossimo carattere è una lettera, quindi leggi una parola.

Parola stringa = ""; // Questa sarà la parola che viene letta.

while (true) {

word + = TextIO.getAnyChar (); // Aggiungi la lettera alla parola.

ch = TextIO.peek (); // Guarda il prossimo personaggio.

if (ch == '\' ') {

// Il prossimo personaggio è un apostrofo. Leggilo e

// se il seguente carattere è una lettera, aggiungi entrambi i simboli

// apostrofo e la lettera sulla parola e continua

// leggendo la parola. Se il personaggio dopo l'apostrofo

// non è una lettera, la parola è fatta, quindi esci dal ciclo.

TextIO.getAnyChar (); // Leggi l'apostrofo.

ch = TextIO.peek (); // Guarda il char che segue l'apostrofo.

if (Character.isLetter (ch)) {

word + = "\" "+ TextIO.getAnyChar ();

ch = TextIO.peek (); // Guarda il prossimo char.

}

altro

rompere;

}

if (! Character.isLetter (ch)) {

// Se il prossimo carattere non è una lettera, la parola è

// finito, quindi rompere il ciclo.

rompere;

}

// Se non abbiamo interrotto il ciclo, il prossimo carattere è una lettera.

}

return word; // Restituisce la parola che è stata letta.

}

Si noti che questo metodo restituirà null quando il file è stato interamente letto. È possibile utilizzare questo come segnale per interrompere l'elaborazione del file di input.

[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex7-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhiXSsGUY_Fdhd_oDBNytGTTqPo-aw)

**Esercizio 7.8:**

Il gioco di Go Moku (noto anche come Pente o Five Stones) è simile a Tic-Tac-Toe, tranne per il fatto che è giocato su una scacchiera molto più grande e l'obiettivo è di ottenere cinque quadrati di fila anziché tre. I giocatori a turno mettono i pezzi su una tavola. Un pezzo può essere posizionato in qualsiasi quadrato vuoto. Il primo giocatore a ottenere cinque pezzi di fila - orizzontalmente, verticalmente o diagonalmente - vince. Se tutti i quadrati vengono riempiti prima che un giocatore vinca, la partita è patta. Scrivi un programma che permetta a due giocatori di giocare a Go Moku l'uno contro l'altro.

Il tuo programma sarà più semplice del programma *Checkers* dalla [sottosezione 7.5.3](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/s5.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhg9f4_L-haJwfA4omDciOlXPuQoSQ#arrays.5.3) . Il gioco si alterna strettamente tra i due giocatori e non è necessario evidenziare le mosse legali. Avrai bisogno solo di due classi, una lezione di cortometraggi per configurare l'interfaccia e una classe di bordo per disegnare la scacchiera e fare tutto il lavoro del gioco. Ciononostante, probabilmente vorrai esaminare il codice sorgente per il programma di pedine, [*Checkers.java*](http://math.hws.edu/javanotes/source/chapter7/Checkers.java) , per idee sullo schema generale del programma.

La parte più difficile del programma è controllare se la mossa che fa un giocatore è una mossa vincente. Per fare ciò, devi cercare in ognuna delle quattro direzioni possibili dal quadrato in cui l'utente ha posizionato un pezzo. Devi contare quanti pezzi ha quel giocatore in fila in quella direzione. Se il numero è cinque o più in qualsiasi direzione, vince quel giocatore. Come suggerimento, ecco parte del codice del mio programma. Questo codice conta il numero di pezzi che l'utente ha in una riga in una direzione specificata. La direzione è specificata da due numeri interi, dirX e dirY .I valori di queste variabili sono 0, 1 o -1 e almeno uno di essi è diverso da zero. Ad esempio, per guardare nella direzione orizzontale, dirX è 1 e dirY è 0.

int ct = 1; // Numero di pezzi in fila appartenenti al giocatore.

int r, c; // Una riga e una colonna da esaminare

r = row + dirX; // Guarda il quadrato nella direzione specificata.

c = col + dirY;

while (r> = 0 && r <13 && c> = 0 && c <13

&& board [r] [c] == player) {

// Square è sulla lavagna, e it

// contiene uno dei pezzi dei giocatori.

ct ++;

r + = dirX; // Vai al prossimo quadrato in questa direzione.

c + = dirY;

}

r = row - dirX; // Ora, guarda nella direzione opposta.

c = col - dirY;

while (r> = 0 && r <13 && c> = 0 && c <13

&& board [r] [c] == player) {

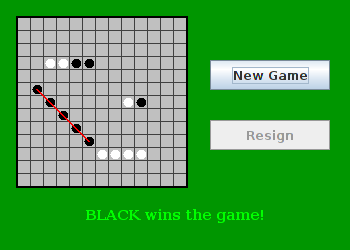
ct ++;

r - = dirX; // Vai al prossimo quadrato in questa direzione.

c - = dirY;

}

Ecco una foto del mio programma, subito dopo che il nero ha vinto la partita.



[Vedi la soluzione](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&ie=UTF8&prev=_t&rurl=translate.google.it&sl=en&sp=nmt4&tl=it&u=http://math.hws.edu/javanotes/c7/ex8-ans.html&xid=17259,15700002,15700022,15700124,15700149,15700168,15700173,15700186,15700201&usg=ALkJrhhZJ6UtOAergZ2tLZs9WXtVGS0z9w)