Esercitazioni di Laboratorio

7. Laboratorio 7

Quiz di autovalutazione

E' aperto il quiz di autovalutazione n.3 (durata 14 minuti). Restera' aperto fino al 3/12.

Programmazione di classi (ripasso per gli esercizi!)

Per programmare una classe è bene seguire le seguenti fasi:

1/ Progettare la classe

La progettazione va fatta prima di cominciare a scrivere codice!

1.1 Processo di astrazione

Stabilire quali sono le caratteristiche essenziali degli oggetti della classe, e fare un elenco delle operazioni che sara` possibile compiere su di essi.

1.1 Definire l'interfaccia pubblica
Definire i costruttori e i metodi da realizzare e per ciascuno di essi scrivere l'intestazione (specificatore di accesso, tipo di valore restituito, nome del metodo, eventuali parametri espliciti).
1.2 Definire le variabili di esemplare
E' necessario individuare le variabili di esemplare (***quante***? ***quali***? dipende dalla classe!) ed eventuali costanti. Per ciascuna variable si deve, poi, definire ***tipo*** e ***nome***.

2/ Scrivere il codice

2.1 Verificate le impostazioni del vostro editor di testi, al fine rendere piu` efficiente il vostro lavoro di programmazione.
L'editor deve essere impostato nel seguente modo:
- tabulazione pari a 3 caratteri;
- indentazione automatica pari a 3 caratteri;
- conversione tabulazione in caratteri;
- numero di riga evidenziato.

2.2 Definire le costanti e le variabili di esemplare
Dopo aver definito le costanti e le variabili di esemplare, compilare e correggere eventuali errori.

2.3 Scrivere l'intestazione dei costruttori e dei metodi pubblici
Scrivere l'intestazione dei costruttori e dei metodi pubblici seguita dal corpo vuoto, come nell'esempio che segue:

```
public double getBalance()
{
}
```

Se un metodo restituisce un tipo di dati, scrivere nel corpo la sola istruzione ***return*** seguita da:
- false se il metodo restituisce un valore di tipo **boolean**
- 'a' se il metodo restituisce un valore di tipo **char**
- 0 se il metodo restituisce un valore numerico (**byte**, **short**, **int**, **long**, **float**, **double**)
- null se il metodo restituisce un riferimento a un oggetto

E' importante programmare l'istruzione ***return*** perche' questo permette di compilare senza errori i metodi prima di scriverne il codice.
Quando si scrive l'effettivo codice del metodo, si modifica l'istruzione ***return*** secondo quanto richiesto dal metodo.
Se il metodo non restituisce un tipo di dati non serve scrivere nulla.

Dopo aver scritto l'intestazione dei metodi, si compili e si correggano eventuali errori.

2.4 Realizzare i metodi
Non appena si è realizzato un metodo, si deve compilare e correggere gli errori di compilazione. Se il metodo e' complicato, compilare anche prima di terminare il metodo.
NON ASPETTARE DI AVER CODIFICATO TUTTA LA CLASSE PER COMPILARE!!!
Se fate questo, vi troverere, probabilmente, a dover affrontare un numero elevato di errori, col rischio di non riuscire a venirne a capo in un tempo ragionevole (come quello a disposizione per la prova d'esame).
SE LA COMPILAZIONE GENERA PIU' DI UN ERRORE, CORREGGERE SOLO IL PRIMO E POI RICOMPILARE.
Un unico errore puo' causare, infatti, piu' messaggi di errore diversi e, quindi la correzione del primo errore puo' far scomparire piu' messaggi di errore, non solo il primo.

3/ Collaudare la classe

Scrivere una *classe di collaudo* contenente un metodo main, all'interno del quale vengono definiti e manipolati oggetti appartenenti alla classe da collaudare.

- E` possibile scrivere ciascuna classe in un file diverso. In tal caso, ciascun file avra' il nome della rispettiva classe ed avra' l'estensione .java. Tutti i file vanno tenuti nella stessa cartella, tutti i file vanno compilati separatamente, solo la classe di collaudo (contenente il metodo main) va eseguita.
- E` possibile scrivere tutte le classi in un unico file. In tal caso, il file .java deve contenere una sola classe public. In particolare, la classe contenente il metodo main deve essere public mentre la classe (o le classi) da collaudare non deve essere public (non serve scrivere private, semplicemente non si indica l'attributo public). Il file .java deve avere il nome della classe public

Esercizio 1

Argomento: classi, tipo booleano, variabili statiche

Scrivere la classe Interruttore che permetta di creare degli oggetti che rappresentano degli interruttori. Ogni interruttore puo' assumere due stati, con il significato che l'interruttore sia su' o giu' (up/down nella descrizione testuale da riportare in output ma gestibile con una variabile booleana).

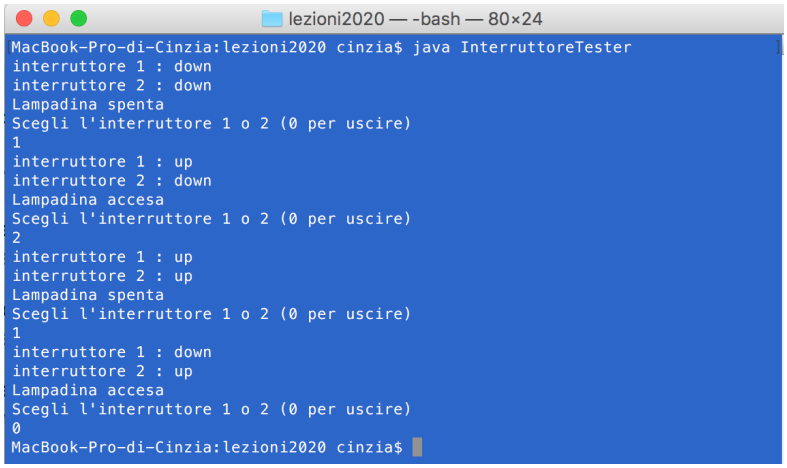
*Immaginate che tutti gli interruttori (ovvero gli oggetti che sono creati dalla classe) siano collegati ad una stessa lampadina regolandone l'accensione e spegnimento. Ogni volta che un interruttore cambia stato, anche la lampadina cambia stato (accesa-> spenta, spenta->accesa). Lo stato della lampadina quindi e' **condiviso** tra tutti gli oggetti interruttore.*

La classe deve contenere: tutte le variabili d'istanza e le variabili statiche ritenute necessarie a descriverne lo stato e i seguenti costruttori e metodi:

```
// Costruttore: inizializza l'interruttore in stato "down"
// Assumiamo "down" corrisponda a false e "up" a true
public Interruttore();

// Metodo di accesso della variabile di esemplare interruttore
public boolean getStatusInterruttore();

//Metodo di accesso alla variabile statica lampadina
public boolean isBulbOn();

//Modificatore: cambia lo stato dell'interruttore
//(e della lampadina!)
public void changeStatus();

//Stampa lo stato dell'interruttore: up/down a seconda
// che status sia true o false
public String printStatus();
```

Per testare la classe, scrivere un programma TestInterruttore che crea due interruttori (oggetti della classe Interruttore) e poi, in maniera iterativa, offre all'utente la possibilita' di agire su uno dei due interruttori (visualizzando l'esito dell'operazione) a seconda che venga inserito il numero 1 o 2, o di terminare l'esecuzione se l'input e' 0.

Esempio:



La classe deve funzionare con 3 o piu' interuttori, dove non e' possibile usare AND e OR per determinare lo stato della lampadina

N.B. Lo scopo dell'esercizio non e' quello di giocare con la logica booleana e/o comprendere la differenza tra interuttori, deviatori e invertitori elettrici, ma bensi' ***esercitarsi nell'uso di variabili statiche (lampadina) e di instanza (interuttore)****.*

Esercizio 2

Argomento: gestione delle eccezioni; uso della terminazione di input; reindirizzamento in input e output

Scrivere un programma eseguibile **ContaInteri.java** che legga da standard input un numero arbitrario di dati e restituisca a standard output il numero di dati letti che sono valori interi. Utilizzare il metodo nextInt() di Scanner per leggere i dati inseriti. Se il dato inserito e' un intero dovra' essere incrementata una variabile contatore precedentemente definita. Nel caso in cui non si inserisca un intero il metodo lancera' l'eccezione **InputMismatchException** che dovra' essere catturata e gestita. In particolare, al verificarsi dell'eccezione non si dovra' incrementare il contatore ma si dovra' liberare il flusso d'ingresso standard dal dato non valido.

Dopo aver realizzato il programma testarlo con inserimento da tastiera.

Dopo aver visto a lezione il reindirizzamento, testare il programma reindirizzando in input il file contaInteri.txt . Provare poi in entrambi i modi ma reindirizzando l'output nel file numeroInteri.txt. Verificare che il contenuto del file sia corretto con il comando "more numeroInteri.txt" su riga di comando del terminale, oppure aprendo il file numeroInteri.txt con un editor.

N.B. cercare nei JavaDoc il pacakage dell'eccezione InputMismatchException. Potete provare anche ad usare Integer.parseInt(scanner.next()) al posto di scanner.nextInt() e quindi l'eccezione NumberFormatException.

Esercizio 3

Argomento: enumerazione delle parole di una stringa tramite Scanner

Acquisire da standard input una stringa composta da piu' parole tramite un oggetto di tipo Scanner che chiamerete **console** e il metodo **nextLine()**.

Creare un nuovo scanner che chiamerete **scan** utilizzando il costruttore che prende come argomento una stringa e passandogli come argomento la stringa letta da standard input.

Elencare una sotto l'altra le parole che compongono la stringa utilizzando i metodi **hasNext()** e **next()** sull'oggetto Scanner **scan**.

N.B. Lo scopo dell'esercizio e' usare la classe Scanner al di fuori dello standard input, denominata anche tokenizer, e per questo vanno usate due instanze della classe Scanner, una dedicata alla lettura da standard input ed una dedicata alla lettura dalla stringa precedentemente letta da standard input. L'esercizio si differenza da quello del laboratorio precedente in quanto non bisogna usare CTRL+D per terminare il buffer di input.

Esercizio 4

Argomento: lettura da file per righe

Scrivere un programma **Leggi1.java** che legga il file input.txt una riga alla volta e stampi a video il contenuto. Si ricordano qui sotto i passi fondamentali per leggere da file:

```
Utilizzare try-with-resources e non preoccuparsi di chiudere le risorse
try(FileReader r = new FileReader(fileName); Scanner scan = new Scanner(r)){
    Si usa lo scanner con i suoi metodi hasNext, next, nextInt, nextDouble,
    nextLine a seconda delle esigenze
}catch(SomeException e){
    gestisco le eccezioni
}
```

Esercizio 5

Argomento: lettura da file per righe; lettura da file per righe e lettura delle parole di ciascuna riga

Scrivere un programma **Leggi2.java** che legga il file input.txt una riga alla volta e stampi le parole contenute in ciascuna riga a video una sotto l'altra.

Ad esempio con file input.txt:

```
ciao, come stai?
io bene e tu?
```

Stampera' in output:

```
ciao,
come
stai?
io
bene
e
tu?
```

Suggerimento: ricordarsi che lo scanner puo' essere utilizzato anche per estrarre "token" da una stringa! Quindi una volta letta la riga da file e' possibile creare un altro scanner passando come argomento la stringa e usare i metodi di scanner per stampare una parola alla volta

Esercizio 6

Argomento: gestione delle eccezioni; uso della terminazione di input; lettura/scrittura file

Scrivere il programma **Leggi3.java** modificando il programma precedente in modo che i segni di punteggiature (virgola e punto di domanda) vengano considerati come separatori. Per far cio' sara' necessario invocare il metodo *useDelimiter passando come parametro la stringa* ***"[,?\\s]+"***. Il contenuto della stringa viene chiamato "espressione regolare". In sostanza stiamo dicendo di considerare come separatori la virgola, il punto di domanda e tutti gli spazi (\\s) ripetuti almento una volta (il + dopo la parentesi quadra). Le espressioni regolari sono un argomento avanzato che esula dal programma del corso.
Pero' possiamo facilmente creare dei separatori, ad esempio provate a definire come separatore la sola lettera "o" e vedere come si separa la frase.

Esercizio 7

Argomento: Lettura e scrittura di file, gestione delle eccezioni, "tokenizzazione" di stringhe

Scrivere un programma CapsCopier.java che

- Riceva dall'input standard due nomi di file di testo, uno in lettura e uno in scrittura
- Apra in lettura il primo file e ne legga il contenuto
- Crei e apra in scrittura il secondo file
- Copi nel secondo file il contenuto del primo, opportunamente modificato in modo che tutte le parole abbiano la prima lettera maiuscola e le seguenti minuscole

Provare il programma usando il file vispateresa.txt come file di input e creando (ad esempio) il file vispateresa2.txt in output.
Approfondimento: modificare il programma in modo che riconosca come due parole distinte anche quelle separate da un apostrofo. Ad esempio, se il file in lettura contiene le parole

```
LA vispA teresa AVEA tra l'erBETTa
```

al termine dell'esecuzione il secondo file dovra` contenere il testo

```
La Vispa Teresa Avea Tra L'Erbetta
```

Suggerimento importante: studiare la **documentazione di Scanner,** e verificare che usando opportuni metodi è possibile **usare un insieme di caratteri delimitatori diverso da quello di default.**

Esercizio 8

Argomento: ricorsione semplice, argomenti sulla riga di comando, lancio/cattura di eccezioni
Scrivere una classe eseguibile avente il funzionamento seguente:

- se sulla linea di comando vengono forniti due o piu' parametri, oppure nessun parametro, il programma termina con una segnalazione di errore
- altrimenti

1. se il parametro fornito non è un numero intero positivo, il programma termina con una segnalazione di errore
2. se il parametro ricevuto e' un numero intero positivo, il programma visualizza sull'uscita la somma dei primi n numeri interi calcolata con un **algoritmo ricorsivo**.

Esercizio 9

Argomento: ricorsione semplice, argomenti sulla riga di comando, lancio/cattura di eccezioni

Scrivere una classe eseguibile avente il funzionamento seguente:

- se sulla linea di comando vengono forniti più o meno di due parametri, il programma termina con una segnalazione di errore
- altrimenti
  - se uno dei due parametri ricevuti non è un numero intero positivo, il programma termina con una segnalazione di errore
  - se entrambi i parametri ricevuti sono numeri interi positivi, il programma visualizza sull'uscita standard il **M.C.D.** tra i due numeri ricevuti, calcolato con un **algoritmo ricorsivo**.

Si scriva un metodo statico ausiliario, recursiveMCD, invocato dal metodo main per realizzare il comportamento sopra indicato. Tale metodo calcola ricorsivamente il massimo comun divisore (MCD) fra due numeri interi positivi *m* ed *n* (con *m>n*), ricevuti come parametri espliciti, usando il noto ***Algoritmo di Euclide***:

- se *n* è un divisore di *m*, allora *n* è il **M.C.D.** tra *m* ed *n*
- altrimenti, il **M.C.D.** di *m* ed *n* è uguale al **M.C.D.** di *n* e del resto della divisione intera di *m* per *n*

Riferimento soluzioni: *RecMCDTester.java*

Esercizio 10

Argomento: ricorsione doppia
Scrivere una classe eseguibile il cui metodo main

- riceva un numero intero dalla riga di comando, oppure (nel caso in cui non vengano forniti argomenti sulla riga di comando) chieda all'utente un numero intero n
- visualizzi l'n-esimo numero di Fibonacci, calcolato usando un algoritmo iterativo

Scrivere una classe eseguibile **RecFib.java** il cui metodo main

- riceva un numero intero dalla riga di comando, oppure (nel caso in cui non vengano forniti argomenti sulla riga di comando) chieda all'utente un numero intero n
- visualizzi l'n-esimo numero di Fibonacci, calcolato usando un algoritmo ricorsivo

- Scrivere una classe eseguibile **IterFib.java** il cui metodo main

    - riceva un numero intero dalla riga di comando, oppure (nel caso in cui non vengano forniti argomenti sulla riga di comando) chieda all'utente un numero intero n
    - visualizzi l'n-esimo numero di Fibonacci, calcolato usando un algoritmo iterativo
- Si consiglia di scrivere due metodi ausiliari statici, **recursiveFib** e **iterativeFib**, invocati da ciascun metodo **main** della rispettiva classe per realizzare il comportamento sopra indicato. Entrambi i metodi ricevono un parametro n di tipo int e (dopo aver verificato la pre-condizione che n non sia negativo) restituiscono un valore di tipo long che rappresenta l'n-esimo numero **Fib(n)** nella sequenza di Fibonacci.
    - Il metodo **recursiveFib** calcola il valore da restituire usando la ricorsione doppia, implementando direttamente la definizione della serie
    - Il metodo **iterativeFib** deve calcolare il valore da restituire senza usare la ricorsione e senza usare strutture dati di memorizzazione (ossia senza array, ma usando soltanto variabili semplici).

  Nei metodi main invocare System.currentTimeMillis() prima e dopo la chiamata al metodo statico e riportare il tempo di esecuzione. Se i tempi non dovessero essere rilevabili in termini di millisecondi potete utilizzare il metodo System.nanoTime()
-
- Provare a lanciare i due programmi piu' volte (giusto per vedere che i tempi sono simili ma non necessariamente uguali per uno stesso algoritmo e uno stesso n, specie al crescere di n) su input crescente.

- Verificare la differenza nell'andamento dei tempi di esecuzione tra i due algoritmi.

MEMO: La sequenza di Fibonacci è così definita:

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n > 1 \end{cases}$$

Esercizio 11

Argomento: ricorsione semplice, argomenti sulla riga di comando

Scrivere una classe eseguibile che verifica se una stringa, fornita come parametro sulla riga di comando, e` palindroma. La verifica che una stringa sia o meno una palindroma deve essere realizzata con un **algoritmo ricorsivo**.
Si ricordi che una stringa e` una palindrome se e` composta da una sequenza di caratteri (anche non alfabetici) che possa essere letta allo stesso identico modo anche al contrario (es. "radar", "anna", "inni", "xyz%u%zyx").

> ***Attenzione:*** Il programma *NON deve avere alcun costrutto iterativo* (cioe` non deve avere cicli).

Verificare il corretto funzionamento del programma con:

- la stringa "omordotuanuoraoarounautodromo"
- una stringa palindroma di lunghezza pari
- una stringa palindroma di lunghezza dispari
- una stringa non palindroma di lunghezza pari
- una stringa non palindroma di lunghezza dispari
- una stringa di lunghezza unitaria (che è ovviamente palindroma)
- una stringa di lunghezza zero (che è ragionevole definire palindroma); per fornire come parametro sulla riga di comando una stringa di lunghezza zero si indica il parametro ""

Riferimento soluzioni: Palindroma.java

Esercizio 12

Argomento: array riempiti solo in parte, ricerca di valore minimo/massimo in un array, ricorsione semplice

Scrivere una classe eseguibile che

- riceve da riga di comando due numeri interi **dim** e **n**;
- crea un array di dimensione **dim**, contenente numeri interi casuali compresi tra 1 e **n**, e lo visualizza a standard output;
- cerca il valore minimo tra quelli contenuti nell'array, tramite un **algoritmo ricorsivo**.

Si consiglia di scrivere un metodo ricorsivo statico che effettui la ricerca e che restituisca un numero intero >= 0 rappresentante il valore minimo trovato.
Suggerimento: in alternativa al metodo **Math.random()**, si puo` utilizzare il metodo **con firma int nextInt(int k)** della classe **java.util.Random**. Prima di usarlo leggere attentamente l'interfaccia pubblica della classe **java.util.Random**. Provare a creare l'oggetto Random sia con il suo costruttore senza parametri, che con quello che ha come parametro un seed (un intero long). Verificare che nel secondo caso, anche eseguendo il programma piu' volte la sequenza generata con lo stesso seed e' sempre la stessa, mentre con il costruttore senza parametri no.

Art Attack (impegnativo ma...)

In questo esercizio viene chiesto di realizzare delle "opere d'arte" in stile Mondrian
(cliccate qui per degli esempi) utilizzando la ricorsione. Potrete poi fotografarle e condividerle su wooclap
https://app.wooclap.com/BPKPPI/questionnaires/6516d15d5a903dfb8431e920
e scegliere la vostra preferita.

Il programma non e' semplice da scrivere perche' richiedere l'utilizzo di metodi per la grafica, ma un po' di lavoro l'ho fatto io per voi.

In particolare avrete a disposizione una classe **MondrianViewer.java** che non dovete "toccare" ma soltanto compilare ed eseguire. Questa classe apre la vostra "tavolozza" e si occupera' di visualizzare automaticamente il risultato del vostro programma.

Vi verra' messa a disposizione anche una classe **BlockComponent.java** che contiene un metodo **paintComponent(...)** a cui mancano solo i parametri delle chiamate ricorsive e un metodo statico **mondrian(Graphics2D g2, int x, int y, int w, int h)** da completare.

Raccomando **a tutti** di leggere la descrizione del programma qui sotto. Io ho caricato gia' anche la soluzione (MondrianViewerSolved.java e BlockComponentSolved.java) per chi preferisce, per questo esercizio, studiare la soluzione proposta e poi modificare i parametri come descritto in fondo alla descrizione per vedere cosa cambia. A chi vuole invece cimentarsi nell'implementazione delle parti mancanti... buon lavoro!

Per quanto riguarda il metodo **paintComponent(...)**, esso sostanzialmente si limita a dividere casualmente in due in larghezza e in altezza della vostra tavolozza, creando quindi 4 regioni. Nelle chiamate ricorsive dovrete mettere le giuste coordinate x,y e le giuste dimensioni di larghezza e altezza per definire le regioni in alto a sx, in alto a dx, in basso a x e in basso a dx. Osservate il codice gia' scritto perche' potra' essere d'aiuto per quello che dovrete implementare voi.

Per quanto riguarda il **metodo statico mondrian**, in esso sono definite tre costanti:
MINw e **MINh** che definiscono, rispettivamente, la larghezza e l'altezza minima di una regione (inizialmente fissate a 120 ma potete modificarle come meglio credete) e la probabilita' **PROBABILITY** di non suddividere una regione "grande" (inizialmente la probabilita' e' a 10, intendendo che nel 10% dei casi una regione grande non viene suddivisa, nel restante 90% dei casi si').

Viene anche gia' creata una variabile Random per poter gestire le varie scelte lasciate al caso. L'algoritmo funziona cosi':

1) se la regione e' sufficientemente piccola (ovvero la sua larghezza e altezza sono inferiori o uguali ai minimi stabiliti) vi trovate nel **"caso base"** e potete procedere a disegnare il bordo e poi a colorarla seguendo le istruzioni del primo if (dopo averne esplicitato la condizione)

2) se la regione non e' sufficientemente piccola decidete con probabilita' PROBABILITY se interrompere le chiamate ricorsive (dopo aver disegnato un rettangolo bianco) o proseguire

3) se il caso ha deciso che dovete continuare a dividere la regione verificate se
3a) e' solo la larghezza ad essere piccola, nel qual caso dividete a caso l'altezza. Per avere una suddivione ragionevole potete imporre che la divisione avvenga tra 1/3 e 2/3 dell'attuale dimensione di altezza. Richiamate ricorsivamente il metodo mondrian(...) su ciascuna delle due regioni ottenute.
3b) e' solo l'altezza ad essere piccola, nel qual caso dividete a caso la larghezza in modo analogo a quanto descritto sopra.
3c) se sia altezza che larghezza sono grandi dividete entrambe le dimensioni e chiamate ricorsivamente il metodo mondrian(...) sulle 4 regioni ottenute.

Una volta che il programma funziona potete lanciarlo piu' volte per ottenere "quadri" diversi ad ogni esecuzione. Fate uno screenshot se volete tenerlo perche' appena chiudete la finestra con la vostra "opera d'arte" (cosa necessaria per terminare il programma) la perderete e alla prossima esecuzione la variabile Random (e quindi la vostra immagine) sara' diversa!
Potete provare a variare l'assegnazione dei colori, la dimensione della regione "piccola" (i minimi non devono nemmeno essere

necessariamente uguali), la probabilita' di suddividere una regione, etc.

Allego qualche mia "opera d'arte" che mi sono divertita a realizzare preparando questo esercizio!