Laboratorio di Programmazione
Laurea in Bioinformatica

II Semestre

8 giugno 2010

1 JAVA: progettazione di classi

Esercizio S2_12

Si progetti una classe di nome Rettangolo, la quale istanzia oggetti in grado di rappresentare rettangoli mediante i loro valori di base b e altezza h. La classe contenga:

1. un costruttore Rettangolo (double b, double h), il quale costruisce un rettangolo di base b e altezza h
2. un costruttore Rettangolo (double b), il quale costruisce un rettangolo di base e altezza b (ovvero un quadrato)
3. un metodo double area(), che restituisce l'area del rettangolo
4. un metodo double perimetro(), che restituisce il perimetro del rettangolo
5. un metodo double diagonale(), che restituisce la diagonale del rettangolo
6. un metodo double diagonale(), che restituisce la diagonale del rettangolo
7. un metodo String toString(), che restituisce una rappresentazione del rettangolo nel formato

```
 --------------------------
|                          |
|                          |
 --------------------------
```

   ben dimensionata limitatamente all'approssimazione ammessa (per semplicità si consideri l'altezza del prompt uguale al doppio della base)
8. un metodo main che, dato un valore di partenza del perimetro, individua il rettangolo che a parità di perimetro possiede area massima al variare di base e altezza e di conseguenza ne stampa i valori di base, altezza e area nonchè una sua rappresentazione. Per determinare l'area massima il metodo adopererà una procedura numerica che fa una ricerca di massimo partendo da 10000 valori scelti a caso della base compatibili col perimetro dato.

Esercizio S2_13

Si progetti una classe di nome Carte Francesi, i cui oggetti modellano il mazzo di 52 carte francesi. Per modellare il mazzo si consiglia di adoperare un array di interi carte[i] di lunghezza 52, in cui ciascun elemento contiene una carta rappresentata da un numero n che va da 1 a 52. La carta corrispondente a un numero n si ottiene ordinando i quattro semi nell'ordine: cuori, quadri, picche, fiori. Per esempio, i numeri da 1 a 13 corrispondono al seme di cuori, e così via.

La classe contenga:

1. un costruttore CarteFrancesi(), il quale istanzia un mazzo attribuendo un valore a ciascun elemento del campo carte[]
2. un metodo statico private static String toString(int n), utile alla classe per stampare la carta n in un formato esplicativo per l'utente (es.: la carta di valore 25 verrà codificata nella stringa regina di quadri)
3. un metodo public String vediTesta, che visualizza la carta in testa al mazzo
4. un metodo public void spostaTesta(), che sposta in fondo al mazzo la carta presente in testa al mazzo
5. un metodo public void mescola(int volte), che rimescola il mazzo un numero di volte come da parametro dato. Il rimescolamento avvenga nel modo seguente: un mazzetto contenente un numero casuale di carte viene estratto dalla testa del mazzo; il mazzetto è reinserito in coda al mazzo in modo da alternare, partendo dalla coda, una carta presa dal mazzetto con una carta esistente nel mazzo

La classe dovrà far parte di un package di nome myclasses accessibile da ambiente Java.

Si progetti infine la classe UsaCarte contenente un metodo main che, appoggiandosi alle risorse della classe CarteFrancesi contenuta nel package myclasses, simula il gioco delle due carte. Il giocatore che dà le carte rimescoli 20 volte il mazzo secondo la regola di mescolamento fornita dalla classe.

2 JAVA: ereditarietà, gerarchie di classi, astrazione

Esercizio S2_14

Sia data la seguente classe `Felino.java`, parte di un package `felini`:

```
package felini;

public class Felino {
    private String nome;

    public Felino(String nome) {
    this.nome = nome;
    }

    public String getNome() {
    return nome;
    }

    public String specie() {
    return "un felino";
    }

    public String toString() {
    return "sono " + specie() +
        " e mi chiamo " + getNome();
    }

    public double quantoCosta() {
    return 100.0;
    }

    public boolean domestico() {
    return true;
    }

    public final boolean possoComprarloCon(double soldi) {
    return domestico() && soldi >= quantoCosta();
    }

}
```

Si crei una gerarchia di classi costruendo due sottoclassi di `Felino`, la prima chiamata `Gatto` e la seconda `Tigre`, ridefinendo i metodi che è opportuno ridefinire e ereditando gli altri. Con lo stesso criterio, si costruiscano due classi ulteriori, `Siamese` e `Scozzese`, che specializzano ulteriormente la classe `Gatto`. Infine, si scriva una classe `MainFelini.java` che istanzia i seguenti oggetti:

```
Gatto aureliano = new Gatto("Aureliano");
Tigre attila = newTigre("Attila");
Siamese chen = new Siamese("Chen")
Scozzese intosh = new Scozzese("Intosh");
```

e ne invoca i metodi toString.

Esercizio S2_15

Si scriva un package `numeri` al cui interno devono essere scritte:

- la classe `NumeroInBase.java`, che rappresenta un numero naturale in base arbitraria. Gli oggetti di tale classe devono avere:

    1. un costruttore `NumeroInBase(numero,base)` che costruisce il numero indicato nella base indicata. Si dia per scontato che `numero` sia non negativo e che `base` sia fra 2 e 36
    2. un metodo `toString()` che restituisce una stringa che descrive il numero nella sua base di numerazione

3. un metodo `piu(altro)` che restituisce il NumeroInBase che rappresenta la somma del NumeroInBase che esegue il metodo e del NumeroInBase `altro`. Nessuno dei due oggetti deve venire modificato. Il risultato deve essere nella stessa base di numerazione del numero che esegue il metodo

- la classe `Binario.java`, sottoclasse di `NumeroInBase.java`, i cui oggetti devono avere:

  1. un costruttore `Binario(numero)` che costruisce il numero indicato in base 2. Si dia per scontato che `numero` sia non negativo
  2. un metodo `toString()` che restituisce una stringa che descrive il numero in base 2
  3. un metodo `piu(altro)` che restituisce il NumeroInBase che rappresenta la somma del Binario che esegue il metodo e del NumeroInBase `altro`. Nessuno dei due oggetti deve venire modificato. Il risultato deve essere in base 2

- la classe `Esadecimale.java`, sottoclasse di `NumeroInBase.java`, i cui oggetti devono avere:

  1. un costruttore `Esadecimale(numero)` che costruisce il numero indicato in base 16. Si dia per scontato che `numero` sia non negativo
  2. un metodo `toString()` che restituisce una stringa che descrive il numero in base 16
  3. un metodo `piu(altro)` che restituisce il NumeroInBase che rappresenta la somma dell'Esadecimale che esegue il metodo e del NumeroInBase `altro`. Nessuno dei due oggetti deve venire modificato. Il risultato deve essere in base 16

Infine si scriva una classe `MainNumeri.java`, esterna al package `numeri`, il cui metodo main crea 19 in base 4, poi in esadecimale, poi in binario e poi in base 22; quindi li stampa tutti e quattro; quindi stampa 19 in base 22 piú 19 in esadecimale e stampa 19 in binario piú 19 in base 22.

Se tutto è corretto, l'esecuzione del main dovrebbe stampare:
19 in base 4: 103
19 in base 16: 13
19 in base 2: 10011
19 in base 22: j
19 in base 22 più 19 in base 16: 1g
19 in base 2 più 19 in base 22: 100110

Esercizio S2_16

Si crei un package `stream` dentro il quale inserire:

- una classe astratta `Stream.java` che rappresenta una sequenza arbitraria infinita di numeri interi n1,n2,n3,n4,n5,n6.... Gli oggetti di tale sequenza devono fornire i metodi:

  1. `primo()` che restituisce n1
  2. `resto()` che restituisce un nuovo stream ottenuto eliminando dallo stream il primo elemento: n2,n3,n4,n5,n6,...
  3. `toString()` che restituisce una stringa che contiene i primi 100 elementi dello stream, separati da virgola

  Lasciare astratti i metodi che non si sanno implementare

- una sua sottoclasse concreta `MaggioriOUgualiA` che rappresenta lo stream infinito dei numeri maggiori o uguali a una data costante. Gli oggetti di tale classe devono avere un costruttore `MaggioriOUgualiA(costante)` in cui si specifica la costante da cui deve cominciare lo stream

- una classe concreta `Positivi` che rappresenta lo stream infinito dei numeri positivi: 1,2,3,4,5,6.... Di quale altra classe conviene che sia sottoclasse?

Si scriva quindi, fuori dal package, una classe MainStream.java che crea e stampa lo stream dei numeri maggiori o uguali a 100 e quello dei numeri positivi.

3 JAVA: ricorsione

Esercizio S2_17

Si scriva una classe che dato un intero k calcoli, sia in versione iterativa che in versione ricorsiva, il k-mo elemento della seguente serie (serie di Fibonacci): 1,1,2,3,5,8,13,21,... La serie è tale che il primo e il secondo numero sono uguali a 1, mentre ciascun elemento successivo è uguale alla somma dei due numeri che lo precedono.

Esercizio S2_18

Si scriva una classe che si serve di metodi ricorsivi per il calcolo del fattoriale di un numero intero n inserito dall'utente e per il calcolo della potenza m^n, dove m è un secondo numero intero fornito dall'utente.

Esercizio S2_19

Si scriva una classe che si serve di un metodo ricorsivo per la stampa di una stringa fornita dall'utente. Il metodo ricorsivo sia del tipo:

```
public static void stampainvert(String s, int i, ConsoleOutputManager out)
```

dove i parametri passati sono la stringa da stampare, un indice che determina l'indice del carattere da cui iniziare a stampare, e un oggetto di tipo ConsoleOutputManager per la stampa.

Esercizio S2_20

Si scriva una classe che si serve di un metodo ricorsivo per ordinare un array di interi in [0,9] fornito dall'utente. Il metodo ricorsivo implementi la seguente strategia in tre passi (algoritmo "mergesort"):

1. Divide l'array non ordinato in due sotto-array di uguale lunghezza (a meno di uno per lunghezze dispari)
2. Ordina ricorsivamente ognuno dei due sotto-array fino fino a ottenere il caso di lunghezza 1 (in questo caso viene restituito l'array stesso)
3. Ricompone, attraverso un metodo merge di servizio, i due sotto-array ordinati in un unico array ordina

Esercizio S2_21

Si scriva una classe che si serve di un metodo ricorsivo per calcolare il massimo comun divisore tra due numeri non negativi. Il metodo si basi sulla seguente uguaglianza:

$$mcd(x,y) = \begin{cases} x & \text{se } y = 0 \\ mcd(y, x \bmod y) & \text{altrimenti} \end{cases} \quad (1)$$

Esercizio S2_22

Si scriva una classe che si serve di un metodo ricorsivo per calcolare il valore di $f(n)$, dove n è chiesto all'utente e f è la funzione definita come

$$f(n) = \begin{cases} 0 & \text{se } n = 0 \\ n + f(n-1) & \text{se } n > 0 \end{cases} \quad (2)$$

4 Java: gestione delle eccezioni

Esercizio S2_23 (Es. 11.1 del testo)

Si scriva una classe `Divisione` che si serve di un metodo `calcolaQuoziente` per calcolare il risultato della divisione tra due interi immessi dall'utente. Si faccia in modo che il metodo main intercetti l'eccezione sollevata dal metodo `calcolaQuoziente` in caso di divisione per zero.

Esercizio S2_24 (Es. 11.9 del testo)

Nella seguente classe, si individuino tutte le eccezioni che possono essere sollevate e si aggiunga il codice necessario a intercettarle e a fornire all'utente i messaggi d'errore adeguati.

```
import prog.io.*

class SequenzaMcd {
    public static void main(String[] args){

    ConsoleInputManager in = new ConsoleInputManager();
    ConsoleOutputManager out = new ConsoleOutputManager();

    int quanti = in.readInt("Quanti interi vuoi inserire? ");

    int[] numeri = new int[quanti];
    for(int i = 0; i<quanti; i++)
        numeri[i]=in.readInt("Numero? ");
    }

    out.println("Questi i numeri inseriti: ");
    for(int i = 0; i<quanti; i++)
        out.println(numeri[i]);
    }

    if(quanti==2){
        int x=numeri[0];
        int y=numeri[1];
        int resto;

        do{
            resto=x%y;
            x=y;
            y=resto;
        }while (resto!=0);
        out.println("Questo il McD dei due numeri: "+x);
    }
}
```

Esercizio S2_25 (Es. 11.12 del testo)

Si scriva una classe `ContaDispari` che si serve di un metodo `public static int[] leggi()` per l'immissione dei dati di un array, e di un metodo `public static int nDispari(int[] h)` che conta e restituisce il numero di valori dispari nell'array passato come argomento. Nel caso in cui il riferimento h sia `null`, il metodo `nDispari`, deve sollevare una eccezione di tipo `EccezioneArray`, definita nel modo seguente:

```
public class EccezioneArray extends RuntimeException{
    public EccezioneArray(String s){
        super(s)
    }
}
```

Il metodo in cui vengono invocate `leggi` e `nDispari` deve gestire l'eccezione in modo da fornire all'utente un messaggio in caso di eccezione di tipo `EccezioneArray`

5 Java: esercizi di ricapitolazione

Esercizio S2_26

Si progetti una classe di nome Treno, la quale istanzia oggetti in grado di rappresentare la potenza di traino della locomotiva, il numero di vagoni e il peso di un treno. Si assuma che l'oggetto abbia senso solo se il peso del treno non supera la potenza e se il numero di vagoni è inferiore a 10: quest'ultimo campo dovrà essere pubblicamente accessibile a tutte le classi che intendano adoperare le risorse fornite dalla classe Treno. La classe contenga:

1. un costruttore Treno(int p), il quale costruisce un oggetto che modella un treno formato dalla sola locomotiva di potenza p
2. un costruttore Treno(int p, int n, int m), il quale se possibile costruisce un oggetto che modella un treno formato dalla locomotiva di potenza p più n vagoni ciascuno pesante m, altrimenti restituisce la sola locomotiva di potenza p
3. un metodo boolean accodaVagone(int m), che se possibile aggiunge al treno modellato un vagone di massa m, e in tal caso restituisce vero; altrimenti restituisce falso
4. un metodo int sganciaVagone(), che se possibile elimina il vagone di coda del treno modellato e in tal caso restituisce la sua massa; altrimenti restituisce 0
5. un metodo int quantoPesa(), che restituisce il peso del treno modellato
6. un metodo int quantoResta(), che restituisce il peso che può essere ancora aggiunto al treno
7. un metodo int quantiVagoni(), che restituisce il numero dei vagoni che attualmente formano il treno
8. un metodo String toString(), che restituisce il modello del treno nel formato "<p>:m1:m2:...", in cui p è la potenza della locomotiva e ciascun elemento tra caratteri due punti contiene la massa m del vagone i-esimo.

Per modellare i vagoni si consiglia di adoperare un array di interi mass[i], in cui ciascun elemento contiene la massa m del vagone i-esimo. La classe dovrà far parte di un package di nome myclasses accessibile da ambiente Java.

Si scriva infine una classe di nome TestTreno che contenga delle semplici istruzioni per la verifica del corretto funzionamento dei metodi della classe Treno.

Esercizio S2_27

Si scriva una classe di nome UsaTreno contenente un metodo main che, appoggiandosi alle risorse della classe Treno contenuta nel package myclasses, simula la creazione di due treni dotati di locomotive di potenza di traino uguale a 50, a cui sono accodati al più 10 vagoni di peso casuale variabile tra 1 e 10. Il metodo successivamente scambia i vagoni in modo da bilanciare quanto più possibile il peso dei due treni secondo una strategia "greedy", ovvero sganciando tutti i vagoni dai due treni e poi assegnando via via il vagone più pesante al treno in quel momento più leggero.

Esercizio S2_28

Si progetti una classe di nome ZeroUno, la quale istanzia oggetti che modellano successioni formate da zero e uno. La classe mantiene la successione in un campo stringa privato e fornisce:

1. un costruttore che, invocato senza parametri, costruisce un oggetto che rappresenta una successione vuota
2. un costruttore che, ricevuto un parametro s di tipo String, costruisce un oggetto che rappresenta la più grande successione di zero e uno contenuta in s nell'ordinamento dato dalla stringa. Ad es., se s è "0er10rt0rt1ghf10h", il campo zerouno dell'oggetto varrà "0100110"
3. un costruttore che, ricevuto un parametro n di tipo int, costruisce un oggetto che rappresenta una successione lunga n di zero e uno casualmente distribuiti
4. un metodo toString() che restituisce la stringa che rappresenta la successione di zero e uno in cui ogni termine è seguito da virgola: se, ad esempio, la successione è 0100 allora la stringa che la rappresenta sarà "0,1,0,0,"
5. un metodo toArray() che restituisce un array di boolean in cui l'elemento di indice i è true o false a seconda che il termine i-esimo della successione (numerata partendo da zero) sia rispettivamente uguale a uno o zero
6. un metodo somma() che RICORSIVAMENTE somma i termini della successione e restituisce un intero contenente la somma. Se ad esempio la successione è 001100010 allora somma() restituirà il valore 3

La classe inoltre contenga un metodo main() il quale, costruiti due oggetti ZeroUno rispettivamente adoperando una stringa e un intero come parametri per la creazione delle successioni a essi associate, dà successivamente la rappresentazione in stringa e la somma di entrambe le successioni.

Esercizio S2_29

Si progetti una classe di nome GrandiIntPositivi, la quale istanzia oggetti che modellano numeri interi positivi di 20 cifre. La classe mantiene cifre comprese tra 0 e 9 in un campo array privato int[] cifre di dimensione costante MAX posta a 20. La classe fornisce:

1. un costruttore che, invocato senza parametri, costruisce un oggetto che rappresenta lo zero, ovvero inizializza l'array ponendo tutti gli elementi a zero;
2. un costruttore che, ricevuto un parametro di tipo String, costruisce un oggetto che rappresenta il numero contenuto nella stringa. Per semplicita' si assuma che la stringa contenga solo cifre decimali. Inoltre, lo stesso costruttore sia eventualmente in grado di scartare la coda di stringhe lunghe piu' di 20 cifre;
3. un metodo GrandiIntPositivi piu(GrandiIntPositivi g) che restituisce, in forma di un nuovo oggetto, la somma dell'oggetto che esegue il metodo con quello passato come parametro. Se la stessa somma supera le 20 cifre allora restituisce l'oggetto che rappresenta il piu' grande intero positivo di 20 cifre;
4. un metodo GrandiIntPositivi perInt(int a) che restituisce il prodotto dell'oggetto che esegue il metodo per l'intero positivo passato come parametro. Il metodo deve essere implementato in forma RICORSIVA, facendo uso del metodo "piu".
5. un metodo toString() che stampa in forma di stringa lunga 20 caratteri il numero rappresentato dall'oggetto che utilizza il metodo;
6. un metodo boolean equals(GrandiIntPositivi g) che verifica l'uguaglianza tra due oggetti in base al numero rappresentato.

La classe contenga infine un metodo main con semplici istruzioni di test della classe (somma fra due oggetti, prodotto per un intero positivo e stampa).

Esercizio S2_30

Si progetti una classe di nome Numeri, i cui oggetti specificano un array a di stringhe di dimensione 100. Ciascun elemento dell'array contiene una stringa binaria il cui simbolo iniziale è 1, mentre tutti i restanti simboli sono 0.

Dopo avere previsto nella classe le necessarie variabili d'istanza, si progettino i seguenti metodi:

1. public Numeri (String numero), il quale attraverso l'uso di una procedura ricorsiva (che eventualmen- te si appoggi su un metodo ausiliario) assegna agli elementi dell'array stringhe ottenute estraendo via via dal parametro numero sottostringhe secondo il formato previsto. Se, ad esempio, numero = "10011000", allora si avrà a[0]="100", a[1]="1", a[2]="1000", a[3]=null, . . . , a[99]=null.
2. public String toString(), il quale restituisce una stringa formata concatenando gli elementi dell'array (esclusi i riferimenti null) attraverso la sequenza di escape newline. Nell'esempio precedente, toString() restituirà

```
100
1
1000
```

Infine, si dia un metodo main che a partire da una stringa binaria obbligatoriamente iniziata dal simbolo 1 istanzia un oggetto Numeri e, successivamente, stampa l'oggetto.

Esercizio S2_31

Si modifichi la classe CarteFrancesi in modo che sia possibile simulare la distribuzione di una mano di poker ad un unico giocatore. Per modellare la singola mano si consiglia di adoperare un ulteriore array di interi mano[i] di lunghezza 5.

I metodi aggiuntivi siano:

1. un metodo public void daiCarte(), che dà le 5 carte
2. un metodo public String vediCarte(), che visualizza le 5 carte in mano
3. un metodo public void accomodo(int[] cambio), che sostituisce le carte nelle posizioni indicate nel vettore cambio con altrettante carte prese dal mazzo
4. un metodo public String combinazione(), che individua la combinazione finale (coppia, doppia coppia, tris, scala, full, colore, poker, scala reale) .

[SUGGERIMENTO: per individuare la combinazione, è consigliabile ordinare modulo 13 le 5 carte (cioè in base al numero e non al seme) e ricavare un vettore delle differenze. Su questo vettore, si possono facilmente individuare i pattern delle combinazioni.

```
Es:
[#0##] -> coppia              [000#] -> poker
[00##] -> tris                [00#0] -> full
... etc.
```

La classe chiamante , estensione di UsaCarte, dà all'unico giocatore 5 carte dopo aver mescolato il mazzo, gli mostra le 5 carte, gli chiede quante e quali carte vuole cambiare e gli comunica infine la combinazione finale individuata.

Esercizio S2_32

Si progetti una classe di nome `T9` che codifica una stringa nella corrispondente sequenza di tasti digitata su un telefono cellulare. Per esempio, un oggetto di tale classe può rappresentare la sequenza 2426026630821. La classe `T9` fornisce:

1. un costruttore che, invocato con una stringa `s`, costruisce un oggetto che rappresenta i tasti del telefono cellulare da premere per ottenere tale stringa, secondo la seguente convenzione:

$$\begin{array}{ccc} {}_1.?! & {}_2\text{abc} & {}_3\text{def} \\ {}_4\text{ghi} & {}_5\text{jkl} & {}_6\text{mno} \\ {}_7\text{pqrs} & {}_8\text{tuv} & {}_9\text{wxyz} \\ {}_*{}^{altro} & {}_0{}^{[\_]} & \end{array} \tag{3}$$

   Per esempio, costruendo un oggetto `T9` a partire dalla stringa `ciao come va?` esso rappresenterà la sequenza di tasti 2426026630821; si noti che gli spazi sono codificati nel carattere numerico 0 e le lettere maiuscole vanno considerate come *altro*;

2. un metodo `toString()` che restituisce la sequenza di tasti rappresentata dall'oggetto;

3. un metodo `isT9Of(String s)` che restituisce `true` se e solo se `s`, tradotta in sequenza di tasti del telefono, come da tabella sopra, dà la stessa sequenza rappresentata dall'oggetto che esegue il metodo;

4. un metodo `equals(T9 other)` che restituisce `true` se e solo se `other` e l'oggetto che esegue il metodo rappresentano la stessa sequenza di tasti.

Si definisca un metodo main all'interno di una classe denominata `SequenzaPalindroma` il quale, acquisita una stringa in ingresso,

1. trova la sequenza palindroma (ovvero simmetrica rispetto al verso di lettura, come ad esempio 12321 e 4224) più grande ottenuta dalla stringa data in ingresso, a partire dal centro

2. restituisce in uscita la sequenza appena trovata

3. restituisce in uscita la porzione della stringa in ingresso che dà origine alla sequenza appena trovata

Esempi:

- `smagliato` è rappresentato dalla sequenza 762454286. Essendo 24542 la più grande sequenza palindroma a partire dal centro, la sottostringa restituita sarà aglia

- `zia` (942) restituisce i, essendo 4 la più grande sequenza palindroma a partire dal centro

- `tuba` (8822) infine restituisce la stringa vuota, essendo vuota la più grande sequenza palindroma a partire dal centro.