ESERCIZI JAVA

Esercizi sulle Interfacce

Esercizio 1:

Java mette a disposizione un'interfaccia chiamata *Comparable*. Quando un oggetto implementa questa interfaccia esso può implementare un metodo chiamato *compareTo* che restituisce un valore intero a variazione dei seguenti casi:
>0 se l'attuale valore è maggiore di quello passatogli come parametro
=0 se l'attuale valore è uguale a quello passatogli come parametro
<0 se l'attuale valore è minore di quello passatogli come parametro

Si vuole dunque scrivere una classe *Quadrato* che implementi i seguenti metodi:

- *public int getLato() //restituisce il valore del lato del quadrato*
- *public void setLato(int lato) //riscrive il valore del lato del quadrato*

la classe *Quadrato* deve inoltre implementare l'interfaccia *Comparable*, e con essa, il metodo *compareTo(Object q).*
Il metodo *compareTo* nella classe *Quadrato* comparerà i lati dei quadrati, e restituirà un valore intero a variazione dei casi come detto precedentemente.
Scrivere il costruttore della classe *Quadrato* a proprio piacimento.

Esercizio 2:

Scrivere una classe *Punto* con le seguenti caratteristiche:

- *public Punto(int x, int y) //costruttore*
- *public int getX() //restituisce la coordinata X*
- *public int getY() //restituisce la coordinata Y*
- *public boolean isOrigin() //restituisce true se il punto è nella coordinata 0,0 //altrimenti restituisce false*
- *public double distance(Punto p) //restituisce la distanza fra i due punti*
- *public String toString() //stampa il punto nella forma P(x,y)*

Scrivere dunque una classe *Cerchio* con le seguenti caratteristiche:

- *public Punto getCenter() //restituisce il punto del centro della circonferenza*
- *public double getRadius() //restituisce il raggio della circonferenza*
- *public double getDiameter() //restituisce il diametro della circonferenza*

inoltre implementare nella classe *Cerchio* l'interfaccia *Comparable* che compara le circonferenza in base al loro raggio.
Scrivere il costruttore della classe *Cerchio* a proprio piacimento.

Esercizio 3:

Scrivere un'interfaccia *Polygon* con i seguenti metodi:

- *ArrayList<Punto> getVertexes() //restituisce i vertici del poligono*
- *double area() // restituisce l'area del poligono*
- *double perimeter() // restituisce l'area del perimetro*

Ricordarsi di importare la il pacchetto *java.util.** per l'uso delle *ArrayList*.
Per la classe *Punto* fare riferimento alla classe *Punto* dell'Esercizio 2.

Esercizio 4:

Basandosi sulla classe *Punto* dell'Esercizio 2 e sull'interfaccia *Polygon* dell'Esercizio 3 scrivere le seguenti classi che implementano l'interfaccia *Polygon*:

classe *Square*:
- *public Square(Punto p1, Punto p2, Punto p3, Punto p4) //costruttore*
- *public double getSide() //ritorna il lato del quadrato*
- *public double getDiagonal() //ritorna la diagonale del quadrato*

classe *Rhomb:*
- *public Rhomb(Punto p1, Punto p2, Punto p3, Punto p4) //costruttore*
- *public double getSide() //restituisce il lato del rombo*
- *public double getLongDiagonal() //restituisce la diagonale lunga del rombo*
- *public double getShortDiagonal() //restituisce la diagonale corta del rombo*

classe *Triangle*:
- *public Triangle(Punto p1, Punto p2, Punto p3) //costruttore*
- *public double height() //restituisce l'altezza del rettangolo*
- *public boolean isIsosceles() //restituisce true se il triangolo è isoscele*
- *public boolean isEquilateral() //restituisce true se il triangolo è scaleno*
- *public boolean isScalene() //restituisce true se il triangolo è scaleno*

Esercizio 5:

Usando le classi scritte nell'Esercizio 4 scrivere una classe *Test* che fa le seguenti operazioni:
- Crea un'*ArrayList* di *Polygon* (riempire l'*ArrayList* con poligoni di proprio gusto)
- Mostra le aree dei vari poligoni
- Mostra i perimetri dei vari poligoni
- Mostra la posizione dei vari vertici dei poligoni
- Identifica i vari poligoni (usare l'operatore instanceof)
- Se il poligono è...
  - … un quadrato, stampare la misura della diagonale
  - … un rombo, stampare la misura delle due diagonali
  - … un triangolo, dire se il triangolo è isoscele, equilatero o scaleno

Esercizio 6:

Definire un'interfaccia *Misura* con attributi unità di misura e valore; definire i metodi *equals()* e c*ompareTo()* (attenzione alle unità di misura!).
Creare delle sottoclassi per le misure di lunghezza e peso per poter effettuare confronti più appropriati.
Definire anche dei metodi di conversione tra una unità e l'altra (per esempio tra metri e cm).
(Suggerimento: definire con costanti intere i valori possibili per le unità di misura).
Si ricordano i valori delle seguenti unità di misura...

1 yarda = 0.9144 metri
1 oncia = 28,5 grammi
1 libbrra = 453,6 grammi

Esercizi sulle ArrayList

Esercizio 1:

Data la seguente classe:

```
public class NumeroNaturale {

	private int value;

	public NumeroNaturale(int n){
		this.setValue(n);
	}

	public NumeroNaturale(){
		value=0;
	}

	public int getValue() { return value; }

	public void setValue(int n) {
		value=n;
		if(n<0) value=value*(-1);
	}

	/* Da completare

	public NumeroNaturale massimo(Integer nn){

	}

	public NumeroNaturale minimo(Integer nn){

	}

	*/

}
```

Creare un array di numeri. Da questo array creare un oggetto di tipo *ArrayList<NumeroNatuale>* che riceve oggetti del tipo indicato tra i diamond (i diamond sono i simboli < >) e inserirli nell'*ArrayList*. Quindi scrivere il corpo dei metodi *massimo()* e *minimo()* della classe *NumeroNaturale* che restituiscono il maggiore/minore fra il *NumeroNaturale* attuale e l'*Integer* passato. Se l'*Integer* non è un *NumeroNaturale*, i metodi *massimo()* e *minimo()* restituiscono *null*.

Esercizio 2:

Completare la seguente classe:

```
public class MyStack<E> {

        private ArrayList<E> stack;

        public MyStack<E>() {

                stack = new ArrayList<E>();

        }

        // Aggiunge in testa il valore passato come parametro
        public void push(E elem){

        //...

        }

        // Restituisce e rimuove il valore in testa allo stack
        // Se non ci sono elementi nello stack restituisce null.
        public E pop(){

        //...

        }

        // Restituisce il numero degli elementi presenti nello stack
        public int size(){

        //...

        }

}
```

Dopo avercompletato la classe scrivere una classe *TestStack* con i seguenti metodi:

Dato un numero intero decimale usando la classe *MyStack<E>* tradurre il numero in binario e stamparlo corretto.
Il metodo restituisce un array di interi che saranno le cifre del numero binario.

Data una stringa, usando i metodi push e pop della classe *MyStack<E>* dice se la stringa è palindroma.
La classe di wrapper per il tipo primitivo *char* è *Charset*.
Per ottenere i singoli char della stringa usare il metodo *getCharAt(int index)* della classe *String*.
Se la stringa è palindroma restituisce *true*.

Data una stringa di parentesi tonde aperte e chiuse, usare i metodi push e pop per controllare che il numero di parentesi aperte e chiuse siano equivalenti e nell'ordine giusto.
Si da per certo che la stringa sia fatta di sole parentesi tonde.
Se le parentesi sono nell'ordine giusto restituisce *true*.
Esempio: " (() ())" OK
"(( )) ()) (" ERRATO

Esercizi con sulla creazione di classi e sull'implementazione di metodi privati, pubblici e statici

Per ogni classe che creerete, definire sempre:

- un costruttore che vuole tutti gli attributi come parametri
- un costruttore che non vuole nessun parametro (costruttore di default)
- i metodi getter and setter

Creare una classe Libro con i seguenti attributi:

- *autore*
- *titolo*
- *editore*

Creare una classe *Bit* con i seguenti attributi:

- *bit (che può avere solo valore 0 o 1)*

e con i seguenti metodi:

- *void set() //setta il bit a 1*
- *void reset() //setta il bit a 0*
- *void complemento() //fa il complemento del bit (0 → 1, 1 → 0)*

Creare una classe *Operazione* senza costruttore che implementa i seguenti metodi statici:

- *media(int[] a)*
- *media(ArrayList<Integer> a)*
- *minimo(int a, int b)*
- *minimo(int[] a)*
- *minimo(ArrayList<Integer> a)*
- *massimo(int a, int b)*
- *massimo(int[] a)*
- *massimo(ArrayList<Integer> a)*
- *fibonacci(int n)*

In una classe si possono definire più metodi con lo stesso nome, basta che cambi il tipo del parametro (vedi *overloading*).

Esercizi tratti da: "Java: la programmazione ad oggetti" di Fabrizia Scorzoni.