Lezione 7

giovedì 25 ottobre 2018 15:39

- Una classe può implementare più interfacce

```
public interface I1 {
    void metodox();
}
public interface I2 {
    int metodoy();
    float metodoz(float e);
}

public class MiaClasse implements I1, I2 {
    =
    public void metodox(){
        =
    }
    public int metodoy(){
        =
    }
    public float metodoz(float e){
        =
    }
}

I1 i = new MiaClasse();
i.metodox();
```

- Un'interfaccia può contenere, oltre a dichiarazioni di metodo, anche costanti.
- Le costanti possono essere usate in tutte le classi che implementano l'interfaccia (o usando direttamente il nome dell'interfaccia).

```
public interface Verbose {
    int SILENT=0;
    int NORMAL=1;
    int VERBOSE=2;
    void setVerbosity(int v);
    int getVerbosity();
}
```

{ sono implicitamente public, static, final

```
public class Logger implements Verbose {
    private int level;
    public void setVerbosity(int v){
        level=v;
    }
    public int getVerbosity(){
        return level;
    }
}
```

Possiamo usare direttamente

```
public void log ( String s ){
    if ( level == NORMAL ) {
            ≡
    } else if ( level == Verbose.SICENT) {
            ≡
    }
    ≡
}
}
```

il nome della costante in quanto Logger implementa l'interfaccia Verbose

possibile anche Logger.VERBOSE

- Con le interfacce si può ottenere un comportamento polimorfico

java.lang.Comparable
che prevede un metodo

int compareTo(—)

public static void sort(Object[] v)

↑ della classe java.util.Arrays

Tipi enumerazione

- Nella forma più semplice sono praticamente uguali ai tipi enumerazione del C/C++

enum Colore {VERDE, GIALLO, ROSSO};

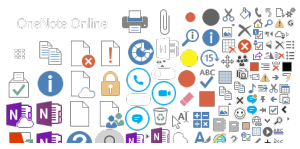
Esempio:

```
class Main {
  enum Colore {VERDE, GIALLO, ROSSO};
  public static void main(String[] args) {

    Colore c1 = Colore.VERDE;
    // Colore c2 = GIALLO; //ERRORE, usare
    Colore.GIALLO

    switch(c1) {
      case VERDE:
        System.out.println("Vai");
        break;
      case GIALLO:
        System.out.println("Rallenta");
        break;
      case ROSSO:
        System.out.println("Fermati");
    }
  }
}
```

PA1819_enum1



- Tutti i tipi enumerazione sono in effetti sottotipi di java.lang.Enum

- Ereditano dei metodi dalla superclasse

public String toString() restituisce il nome dell'enumeratore

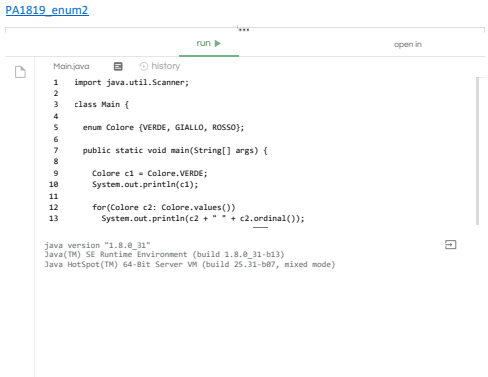
E è il tipo enumerazione

public static E[] values() restituisce un array contenente tutti i valori

public static E valueOf(String s) restituisce l'enumeratore corrispondente alla stringa s (lancia IllegalArgumentException se s non corrisponde a nessun valore)

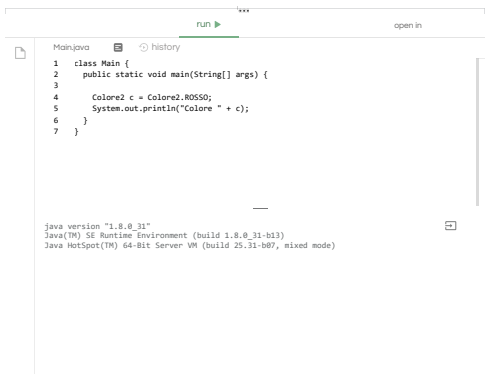
public int ordinal() restituisce il numero d'ordine dell'enumeratore (partendo da zero).

PA1819_enum2



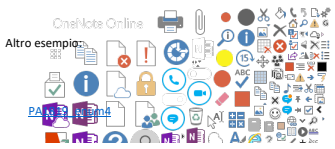
- È possibile ridefinire metodi, aggiungere di nuovi, e aggiungere stato

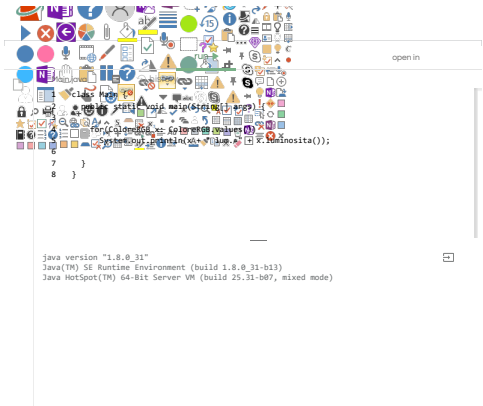
PA1819_enum3



Altro esempio:

PA1819_enum4





Esercizio:

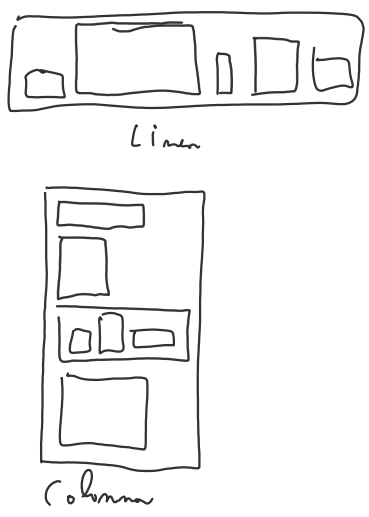
Finestre
Nella costruzione di una interfaccia grafica, gli elementi che compongono l'interfaccia vengono normalmente modellati come finestre rettangolari. Alcune di queste finestre sono visibili (bottoni, aree di testo etc.), mentre altre, strutturali, servono a raggruppare le altre finestre e a disporle sullo schermo (per esempio, allineate orizzontalmente o in colonna).

Supponiamo di definire soltanto i seguenti tipi di finestra
e di essere interessati esclusivamente al calcolo delle loro dimensioni:
• **Bottone**: una finestra visibile, che ha una altezza e una larghezza decisi al momento della creazione.
• **Linea**: una finestra strutturale che contiene altre finestre, disposte orizzontalmente. L'altezza di una Linea è il massimo tra le altezze di tutte le finestre contenute, mentre la sua lunghezza è la somma di tutte le lunghezze delle finestre contenute.
• **Colonna**: una finestra strutturale che contiene altre finestre, disposte verticalmente. L'altezza di una Colonna è la somma di tutte le altezze delle finestre contenute, mentre la sua lunghezza è il massimo tra le lunghezze di tutte le finestre contenute.

Sia ad un oggetto di tipo Linea che ad un oggetto di tipo Colonna è possibile aggiungere una nuova finestra in coda a quelle già contenute, oppure estrarre l'ultima finestra inserita. Realizzare le classi che definiscono i tipi di finestra appena descritti, facendo uso delle seguenti definizioni:

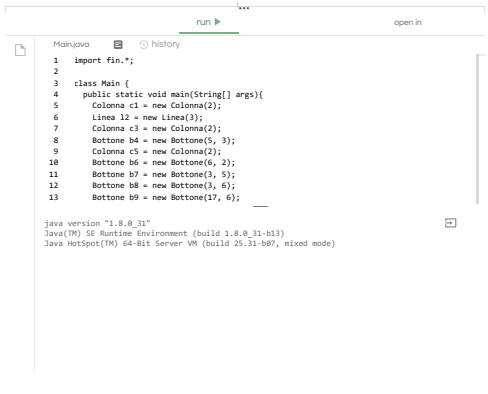
```
package fin;
public class Rettangolo {
        public int larghezza;
        public int altezza;
        public Rettangolo(int larghezza, int altezza) {
                this.larghezza = larghezza;
                this.altezza = altezza;
        }
}

package fin;
public interface Finestra {
        Rettangolo dimensioni();
}
```



Soluzione

PA1819_finestre



Eccezioni

- Le eccezioni permettono di rappresentare situazioni di errore e loro eventuale recupero.
- Separare il codice relativo all'esecuzione normale rispetto a quello per la gestione degli errori

```
if (ERROR_CODE == f()) {
        ≡
}
```

- Esempi di item : :

1) provo a scrivere in un file e non ho il permesso

2) c'è un errore nella comunicazione con un servizio remoto

] è possibile prevedere azioni di recupero

3) accedo a un array usando un indice non valido

4) invoco un metodo su un riferimento null

] Potrei ma non lo faccio, sono errori logici

5) errore in fase di linking] Non gestisco perché non ho strumenti per risolvere il problema.

- Le eccezioni Java possono essere lanciate e catturate.

- Un'eccezione viene lanciata nel punto in cui viene rilevata la situazione di errore
  - il normale flusso di esecuzione viene alterato: la prossima istruzione che verrà eseguita è quella del "primo" gestore in grado di catturare quel tipo di eccezione.

Le eccezioni sono organizzate secondo una gerarchia