UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3
e
Laboratorio di Programmazione 3

Enumerazioni
e
Tipi innestati

Proff. Angelo Ciaramella – Emanuel Di Nardo

Classe innestate

- classe innestata
  - classe definita all'interno di un'altra classe
  - risparmio di codice
    - la classe interna ha accesso ai membri della classe esterna anche se dichiarati `private`
  - possibilità di utilizzare i modificatori d'accesso come per i membri di una classe

Classe innestate

```
public class Outer {
    private String messaggio = "Nella classe ";
    private void stampaMessaggio() {
       System.out.println(messaggio + "Esterna");
    }
    /* la classe interna accede ai membri privati
    della classe che la contiene */
    public class Inner {
       public void metodo() {
            System.out.println(messaggio + "Interna");
       }
       public void chiamaMetodo() {
            stampaMessaggio();
       }
    //. . .
    }
//. . .
}
```

Esempio di classe innestata

Classe innestate

- Proprietà
  - deve avere un identificatore differente dalla classe che la contiene
  - si possono utilizzare tutti i modificatori d'accesso per dichiarare una classe innestata
  - Per istanziare oggetti di una classe innestata (non privata) al di fuori della classe in cui è stata dichiarata

```
Outer outer = new Outer();
Outer.Inner inner;
inner = outer.new Inner();
```

  - Ha accesso sia alle variabili d'istanza sia a quelle statiche della classe in cui è dichiarata

Classe innestate

- Proprietà
  - Una classe innestata *si può dichiarare anche all'interno di un metodo*, viene detta classe locale ("local class")
  - Se viene dichiarata statica diventa automaticamente una "top-level class"
    - non sarà più definibile come classe interna
    - non godrà della proprietà di poter accedere alle variabili d'istanza della classe in cui è definita
  - Solo se dichiarata statica può dichiarare membri statici
  - Può essere dichiarata astratta
  - Nei metodi di una classe interna è possibile utilizzare il reference this
  - Se compiliamo una classe che contiene una classe interna, saranno creati due file

Classi anonime

- Classi anonime
  - Sono classi innestate senza nome
  - Godono delle stesse proprietà delle classi innestate
  - Sono utilizzate per gli stessi scopi

- La dichiarazione richiede
  - contestualmente alla dichiarazione della classe venga anche istanziato un suo oggetto
  - l'esistenza di una sua superclasse o di una sua superinterfaccia di cui sfrutterà il costruttore (virtualmente nel caso di un'interfaccia)

Classi anonime

- una classe anonima viene dichiarata con lo scopo di fare override di uno o più metodi della classe che estende

- anche le classi anonime si possono dichiarare all'interno di metodi
  - accedere alle variabili locali e ai parametri, solo se dichiarati `final`

Classe innestate

```
public class Outer4 {
    private String messaggio = "Nella classe ";
    // Definizione della classe anonima e sua istanza
    ClasseEsistente ce = new ClasseEsistente() {
       @Override
       public void metodo() {
           System.out.println(messaggio+"anonima");
       }
    }; //Si noti il ";"
//. . .
}
// Superclasse della classe anonima
public class ClasseEsistente {
    public void metodo() {
       System.out.println("Nella classe esistente");
    }
}
```

Esempio di classe anonima

Tipi enumerazioni

- Tipi enumerazioni (enumerated types)
  - si possono salvare con il suffisso `.java`

```
public enum MiaEnumerazione {
     UNO, DUE, TRE; // elementi o valori
}
```

costanti statiche

Ereditarietà e polimorfismo

- Un'enumerazione viene trasformata dal compilatore in una classe che estende la classe astratta `Enum`
  - ereditiamo dalla classe `java.lang.Enum` diversi metodi

```
System.out.println(MiaEnumerazione.UNO);
```

`toString()`

- Ogni elemento di `MiaEnumerazione` è di tipo `MiaEnumerazione`
- Non si può estendere o estendere un'altra classe

Interfacce

- È invece possibile far implementare un'interfaccia

```
public interface Numeratore {
    void stampaIndice();
}
```

Interfaccia

```
public enum MiaEnumerazione2 implements Numeratore {
     UNO, DUE, TRE;
     @Override
     public void stampaIndice() {
        System.out.println("Indice: "+this.ordinal());
     }
}
```

Implementazione dell'interfaccia

```
Numeratore n = MiaEnumerazione2.DUE;
n.stampaIndice();
```

Polimorfismo

```
Indice: 1
```

Risultato

Ulteriori informazioni

- E' possibile anche creare
  - variabili
  - metodi
  - costruttori
    - possono essere usati dai valori dell'`enum`
    - overload
    - `private`
  - tipi innestati
    - classi
    - interfacce
    - enumerazioni
    - annotazioni

Ulteriori informazioni

```
public enum MiaEnumerazione3 {
    ZERO(), UNO(1), DUE(2), TRE(3);
    private int valore;
    private MiaEnumerazione3() {
    }
    MiaEnumerazione3(int valore) {
        setValore(valore);
    }
    public void setValore(int valore){
        this.valore = valore;
    }
    public int getValore(){
       return this.valore;
    }
    @Override
    public String toString() {
       return "" + valore;
    }
}
```

Esempio di utilizzo di costruttori.

Enumerazioni innestate

```
public class Volume {
    public enum Livello {ALTO, MEDIO, BASSO};
    // implementazione della classe . . .
    public static void main(String args[]) {
       System.out.println(Livello.ALTO);
    }
}
```

Una `enum` innestata è implicitamente statica

Esercizio

- Creare un'interfaccia CurrencyConverter con un metodo convert:
  - Si occuperà di effettuare le conversioni tra valute
- Creare una classe EURConverter che implementa CurrencyConverter:
  - 1 USD = 0,97 EUR
- In BankAccount creare una variabili tramite classe anonime ed una classe innestata:
  - JPYConverter di tipo CurrencyConverter (classe anonima):
    - 1 USD = 139,96 JPY (yen)
  - GBPConverter di tipo CurrencyConverter (classe innestata):
    - 1 USD = 0,85 GBP (sterlina)
- Inserire un metodo convertTo che capisca tramite enumeration il tipo di conversione da eseguire:
  - enum Currency { . . . }