Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Introduzione agli Enumerativi

Loris Grossi, Fabio Landoni, Andrea Baldassari

Contenuto realizzato in collaborazione con: T. Leidi, A.E. Rizzoli, S. Pedrazzini

Fondamenti di Informatica
Bachelor in Ingegneria Informatica

SA 2022

Obiettivo

Essere in grado di utilizzare, in maniera corretta, gli enumerativi all'interno di programmi Java.

Obiettivi della lezione:

- Conoscere il concetto di enumerativo nella programmazione.
- Saper utilizzare la sintassi degli enumerativi per:
  - creare nuovi tipi di dato e
  - accedere ai valori.
- Conoscere e saper usare le funzioni:
  - `ordinal()`,
  - `valueOf()`,
  - `values()`.
- Conoscere e saper gestire il concetto di esaustività in riferimento agli enumerativi e al nuovo `switch`.

Inventare nuovi tipo di dato

I tipi di dato primitivi (`boolean`, `int`, `double`, `char`, …) sono molto utili ***ma presentano i loro limiti***.

Ad esempio, se volessimo un tipo di dato per i valori di un semaforo (verde, giallo, rosso) come potremmo fare?

Il tipo `boolean` permette solo due valori: `true` e `false`. Quindi non va bene.

Può essere utilizzato il tipo `int`, ma di valori possibili ne concede decisamente troppi.

In alcuni casi è utile poter definire nuovi tipi di dato.

Inventare nuovi tipo di dato

Definire un nuovo tipo di dato vuole dire ***inventare*** un tipo di dato di cui si potranno ***poi dichiarare delle variabili***.

Ad esempio, si vuole specificare il tipo `Semaforo` con valori: rosso, giallo, verde in modo da poter dichiarare la variabile:

```
Semaforo statoSemaforo;
```

Alla quale sarà possibile assegnare i valori:

```
statoSemaforo = Semaforo.ROSSO;
statoSemaforo = Semaforo.GIALLO;
statoSemaforo = Semaforo.VERDE;
```

Inventare nuovi tipo di dato

Come già introdotto, è possibile definire nuovi tipi di dato utilizzando ***le classi***. Discuteremo questa possibilità nel corso del semestre.

Ma è possibile farlo anche utilizzando ***gli enumerativi***.

Enumerativi

Un ***tipo enumerativo*** specifica un ***insieme prefissato di valori*** che possono essere assunti da una variabile di quel tipo. La sintassi da utilizzare per definire un enumerativo è:

```
enum NomeTipoEnum {
     LISTA, DI, VALORI
}
```

Esempio:

```
enum Stagione {
     PRIMAVERA, ESTATE, AUTUNNO, INVERNO
}
```

I tipi enumerativi vanno definiti (per ora) ***nello stesso file del programma*** ma al di fuori (di regola prima) del programma.

Enumerativi: esempio

```
enum Semaforo {
  ROSSO, GIALLO, VERDE
}

public class ProvaSemafori {
  public static void main(String[] args) {
    Semaforo sem1 = Semaforo.VERDE;
    Semaforo sem2 = Semaforo.ROSSO;

    if (sem1 == sem2) {
      System.out.println("Il colore dei semafori è uguale");
    } else {
      System.out.println("Il colore dei semafori è diverso");
    }
  }
}
```

Enumerativi

Ogni ***valore*** di un tipo enumerativo è ***costante***. Non si può cambiare il contenuto fra le parentesi graffe una volta che è stato definito.

Per ***accedere ad un valore*** di un tipo enumerativo ***si usa il*** "***.***":

```
Stagione.AUTUNNO
```

Dato un tipo enumerativo, si possono dichiarare variabili di quel tipo:

```
Stagione raccoltaFunghi;
raccoltaFunghi = Stagione.AUTUNNO;
```

Enumerativi: esempio

```
enum Mese {
    GENNAIO, FEBBRAIO, MARZO, APRILE, MAGGIO, GIUGNO, LUGLIO,
    AGOSTO, SETTEMBRE, OTTOBRE, NOVEMBRE, DICEMBRE
}

public class MesiAnno {
    public static void main(String[] args) {
        int giorno = 31;
        Mese mese = Mese.DICEMBRE;
        int anno = 2017;

        if (giorno == 31 && mese == Mese.DICEMBRE) {
            giorno = 1;
            mese = Mese.GENNAIO;
            anno++;
        }
    }
}
```

Operazioni con gli enumerativi

Gli enumerativi mettono a disposizione diverse ***funzioni*** che permettono di interrogarne le proprietà e di eseguire operazioni sugli stessi:

- ***`v1.ordinal()`*** permette di scoprire la posizione del valore `v1` nella lista di valori di un tipo enumerativo,
- ***`NomeTipoEnum.valueOf(s1)`*** converte la stringa `s1` nel valore corrispondente dell'enumerativo `NomeTipoEnum`,
- ***`NomeTipoEnum.values()`*** restituisce un array contente tutti i valori dell'enumerativo `NomeTipoEnum`.

`ordinal()`

Utilizzando la funzione ***`ordinal()`*** è possibile scoprire la posizione di un valore nella lista di valori di un tipo enumerativo.

La posizione ***parte da 0***.

Esempio:

```
Stagione.ESTATE.ordinal()
```

è pari a `1` (si parte da 0).

```
Stagione stagioneFredda = Stagione.INVERNO;
System.out.println(stagioneFredda.ordinal());
```

stamperà a schermo `3`.

Esempio

```
enum Mese {
  GENNAIO, FEBBRAIO, MARZO, APRILE, MAGGIO, GIUGNO, LUGLIO, AGOSTO, SETTEMBRE,
  OTTOBRE, NOVEMBRE, DICEMBRE
}

enum Giorno {
  LUNEDI, MARTEDI, MERCOLEDI, GIOVEDI, VENERDI, SABATO, DOMENICA
}

public class GiorniEMesi {
  public static void main(String[] args) {
    Giorno giorno = Giorno.VENERDI;
    Mese mese = Mese.OTTOBRE;

    System.out.print("Sono della bilancia poiché sono nato in " + mese + "; ");
    System.out.println("è il " + (mese.ordinal() + 1) + "° mese dell'anno.");

    System.out.print("La settimana lavorativa termina al " + giorno + ". ");
    System.out.println(giorno + " è il " + (giorno.ordinal() + 1)
            + "° giorno della settimana.");
  }
}
```

Conversione di stringhe in tipi enumerativi e viceversa

Per ogni tipo `enum` dichiarato esiste la routine ***`valueOf()`*** che converte la stringa nel valore corrispondente di un enumerativo. Segnala l'errore `IllegalArgumentException` se la stringa non corrisponde a nessun valore.

```
enum SemeCarta{
    CUORI, QUADRI, FIORI, PICCHE
}

// ...
SemeCarta seme = SemeCarta.valueOf("FIORI");
```

Viceversa, un dato di tipo `enum` può essere concatenato con una stringa o utilizzato direttamente con i comandi per l'output su schermo:

```
SemeCarta seme = SemeCarta.CUORI;
System.out.println("Seme carta: " + seme);
```

Tutti i valori di un enumerativo

Per ogni tipo `enum` dichiarato esiste la routine ***`values()`*** che restituisce un array contente tutti i valori dell'enumerativo.

```
enum Giorno {
    LUNEDI, MARTEDI, MERCOLEDI, GIOVEDI, VENERDI,
    SABATO, DOMENICA
}

public class TestGiorni {
    public static void main(String[] args) {
        System.out.print("Il secondo giorno è: ");
        System.out.println(Giorno.values()[1]);

        for (Giorno giorno : Giorno.values()) {
            System.out.print(giorno + " è il giorno numero ");
            System.out.println(giorno.ordinal() + 1);
        }
    }
}
```

Enumerativi e istruzione `switch` tradizionale

```
enum Stagione { PRIMAVERA, ESTATE, AUTUNNO, INVERNO }

public class Stagioni {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Inserisci una stagione: ");
    Stagione stagione = Stagione.valueOf(input.nextLine().toUpperCase());
    switch (stagione) {
    case PRIMAVERA: // Importante: non Stagione.PRIMAVERA!!
      System.out.println("Marzo, Aprile, Maggio");
      break;
    case ESTATE:
      System.out.println("Giugno, Luglio, Agosto");
      break;
    case AUTUNNO:
      System.out.println("Settembre, Ottobre, Novembre");
      break;
    case INVERNO:
      System.out.println("Dicembre, Gennaio, Febbraio");
      break;
    }
    input.close();
  }
}
```

Enumerativi e istruzione nuovo `switch`

```
enum Stagione { PRIMAVERA, ESTATE, AUTUNNO, INVERNO }

public class Stagioni {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Inserisci una stagione: ");
    Stagione stagione = Stagione.valueOf(input.nextLine().toUpperCase());
    switch (stagione) {
    // Importante: non Stagione.PRIMAVERA!!
    case PRIMAVERA -> System.out.println("Marzo, Aprile, Maggio");
    case ESTATE -> System.out.println("Giugno, Luglio, Agosto");
    case AUTUNNO -> System.out.println("Settembre, Ottobre, Novembre");
    case INVERNO -> System.out.println("Dicembre, Gennaio, Febbraio");
    }
    input.close();
  }
}
```

Enumerativi, nuovo `switch` ed esaustività

Il nuovo `switch` ***usato come espressione*** (ossia un costrutto che ritorna un valore) ha la caratteristica di ***esaustività***: il codice non è compilabile nel caso in cui non siano stati contemplati tutti i casi.

Per esempio, nel caso dell'enumerativo

```
enum Stagione { PRIMAVERA, ESTATE, AUTUNNO, INVERNO }
```

il codice

```
String mesi = switch (stagione) {
case PRIMAVERA -> "Marzo, Aprile, Maggio";
case ESTATE -> "Giugno, Luglio, Agosto";
case AUTUNNO -> "Settembre, Ottobre, Novembre";
};
```

non compila poiché manca la gestione del valore `INVERNO`.

L'errore visualizzato è:

A Switch expression should cover all possible values

Enumerativi, nuovo `switch` ed esaustività

Per garantire la caratteristica di ***esaustività nel caso degli enumerativi*** si hanno due possibilità:

- definire una clausola ***`case` per ogni valore*** dell'enumerativo,
- oppure definire la clausola ***`default`***.

È ***preferibile*** definire una clausola ***`case` per ogni valore*** dell'enumerativo. In questo modo se all'enumerativo vengono aggiunti nuovi valori, non ci si dimentica di gestire il comportamento dello `switch` per questi nuovi valori. Il compilatore ci avvertirebbe infatti che non sono stati gestiti tutti i casi possibili.

Con l'uso della clausola `default` invece, il compilatore non segnala nessun errore e potremmo correre il rischio di dimenticarci di specificare i nuovi casi. Questo potrebbe portare ad un codice mal funzionante se il comportamento desiderato non è quello del caso di default.

Riepilogo

- Dichiarazione di un tipo enumerativo
- Utilizzo di un tipo enumerativo
- La funzione `ordinal()`
- La funzione `valueOf()`
- La funzione `values()`
- Enumerativi e `switch` tradizionale
- Enumerativi e nuovo `switch`
- Esaustività