

Programmazione 2

Sesta esercitazione

26/04/2024

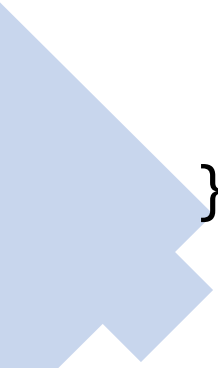


Obiettivi dell'esercitazione

1. Enumerazioni `enum`
2. Costrutto `switch-case`
3. Generics



Problema: voglio rappresentare i giorni della settimana

```
public class GiornoDellaSettimana {  
    public final int giorno;  
  
    public GiornoDellaSettimana(int giorno) {  
        if (giorno >= 0 && giorno <= 6) {  
            this.giorno = giorno;  
        } else { ... }  
    }  
  
    public boolean festivo() {  
        return this.giorno > 4;  
    }  
}
```



enum e switch-case: niente di nuovo rispetto a C

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

```
boolean festivo(GiornoDellaSettimana giorno) {  
    switch (giorno) {  
        case SABATO: case DOMENICA:  
            return true;  
        default:  
            return false;  
    }  
}
```

`switch-case`: quasi niente di nuovo rispetto a C

- Tipi primitivi

`byte`, `short`, `char`, `int`, rispettive classi wrapper

- `String` (Java ≥ 7)

Viene invocato `String.equals(String)`

- Enum

- Comportamento "a cascata": una volta trovato il case giusto, va avanti fino al primo `break`

enum: qualcosa di nuovo

In Java, "sotto il cofano" hanno sottoclassi di `java.lang.Enum`

Nota: non possiamo estenderla direttamente

```
public enum CittàBelle {  
    CAGLIARI, ...  
}
```

// Equivale concettualmente a:

```
public class CittàBelle extends Enum {  
    public static final CittàBelle CAGLIARI =  
        new CittàBelle(); ...  
}
```

enum: qualcosa di nuovo

- Essendo delle classi, possiamo definire
 - Metodi
 - Variabili
 - Costruttori
- Inoltre, il compilatore aggiunge per noi dei metodi di utilità, tra cui:
 - Il metodo statico `values()` rende un vettore con tutte le nostre costanti
 - Il metodo `toString()` rende, per una costante, il suo nome
- **Nota sui costruttori:** non li possiamo invocare direttamente, si occupa Java di invocarli per allocare le costanti che abbiamo definito

Enum: qualcosa di nuovo

```
public enum CittàBelle {  
  
    CAGLIARI ("Sardegna"),  
    VENEZIA ("Veneto");  
  
    private String regione;  
  
    CittàBelle(String regione) {  
        this.regione = regione;  
    }  
}
```


Enum: esercizio (1/2)

Vogliamo definire un'enumerazione `Mesi` per i mesi dell'anno

- Ogni costante dell'enumerazione `Mesi` ha come proprietà:
 - Il numero di giorni
 - La stagione
- Ogni costante dell'enumerazione `Mesi` ha i metodi getter:
 - `getGiorni()`
 - `getStagione()`

Enum: esercizio (2/2)

Vogliamo una classe `TestMesi` che per ogni mese stampi:

- Il nome (con la sola iniziale Maiuscola)
 - Il numero dei giorni
 - La stagione
-
- Ad esempio, vogliamo che l'output del programma sia:
A Gennaio ci sono 31 giorni ed è inverno.
...
A Giugno ci sono 30 giorni ed è estate.
...

Tipi generici

Talvolta, dobbiamo lavorare su strutture dati che possano accettare oggetti di diversi tipi

- Liste
- Stack
- Nodi di BST (*alberi binari di ricerca*)
- ...

Java mette a nostra disposizione i tipi generici

Utili nelle Classi o nelle interfacce in cui il tipo trattato è un parametro della classe stessa.

Tipi generici: che vantaggio? Posso usare Object!

```
public class Box {  
    private Object contenuto;  
  
    Box(Object o) { this.contenuto = o; }  
  
    public Object get() { return this.contenuto; }  
}
```

...

```
Box numeroInBox = new Box(1);  
Box booleanoInBox = new Box(false);
```

Tipi generici: che vantaggio? Posso usare Object!

```
Box numeroInBox = new Box(1);  
Box booleanoInBox = new Box(false);  
...  
Integer successivo = (Integer) numeroInBox.get();  
successivo += 1;  
Boolean complemento = (Boolean) booleanoInBox.get();  
complemento = !complemento;  
...  
complemento = (Boolean) numeroInBox.get();
```

Tipi generici: che vantaggio? Posso usare Object!

```
$ javac Box.java  
$ java Box
```

```
Exception in thread "main"  
java.lang.ClassCastException: java.lang.Integer cannot  
be cast to java.lang.Boolean  
    at Box.main(Box.java:18)
```

**Non solo devo ricordarmi di usare i cast, ma se sbaglio qualcosa lo
scopro solo al momento dell'esecuzione!**

Tipi generici: molto meglio di Object

```
public class Box<T> {  
    private T contenuto;  
  
    Box(T contenuto) { this.contenuto = contenuto; }  
  
    public T get() { return this.contenuto; }  
}  
  
...  
Box<Integer> numeroInBox = new Box<Integer>(1);  
Box<Boolean> booleanoInBox = new Box<Boolean>(false);
```

Tipi generici: molto meglio di Object

```
Box<Integer> numeroInBox = new Box<Integer>(1);  
Box<Boolean> booleanoInBox = new Box<Boolean>(false);  
...  
Integer successivo = numeroInBox.get();  
successivo += 1;  
Boolean complemento = booleanoInBox.get();  
complemento = !complemento;  
...  
complemento = numeroInBox.get();
```


Tipi generici: molto meglio di Object

```
$ javac Box.java
```

```
Box.java:18: error: incompatible types: Integer cannot  
be converted to Boolean  
        complemento = numeroInBox.get();
```

Molto meglio scoprirlo al momento della compilazione!

Ripasso: tipi generici

- Quando usiamo un'interfaccia come tipo di una variabile, possiamo referenziare oggetti che implementano quella interfaccia

- Anche le interfacce possono avere parametri di tipo

```
public interface List<E>
```

- Quando l'interfaccia ha parametro di tipo, il parametro di tipo degli oggetti deve essere compatibile

```
List<Integer> numeri = new ArrayList<Integer>();  
numeri.add(20);  
Integer i = numeri.get(0);
```

Ripasso: tipi generici

- Convenzioni sulla lettera usata per indicare il parametro:
 - **T** è la scelta principale
 - **S, U, V**, eccetera se ci sono più tipi
 - **N** indica che si useranno numeri
 - **K** e **V** sono spesso usati per indicare la coppia di tipi per chiave e valore
 - **E** indica un elemento quando stiamo definendo una generica collezione

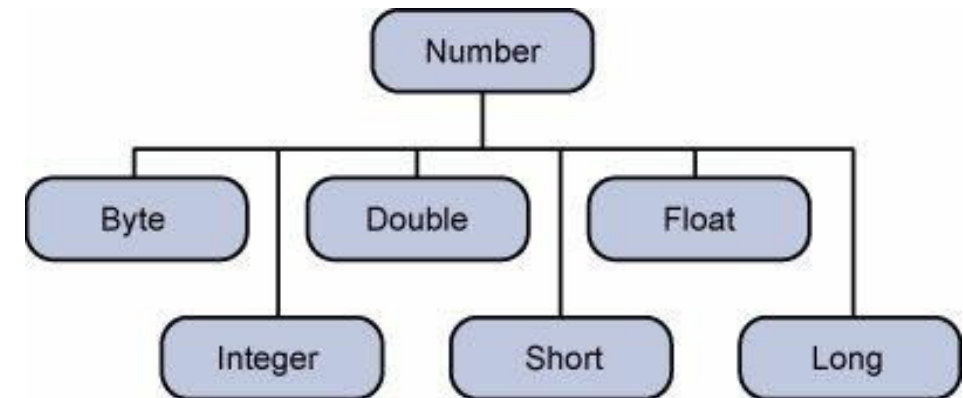
Ripasso: tipi generici

Può essere necessario restringere il tipo passato come argomento

Esempio: una classe che lavora solo con tipi numerici

```
public class Point<T extends Number> {  
    private T x;  
    private T y;  
    public T getX() { return this.x; }  
}
```

```
Point<Double> p = new Point<>();
```



Nel caso si debba specificare un'interfaccia si usa la keyword `implements`

FINE ESERCITAZIONE