Programmazione Orientata agli Oggetti

Riflessione

Introduzione

- La riflessione (o *introspezione*) è una interessante caratteristica del linguaggio Java
- Permette di scrivere codice la cui funzionalità principale consiste nell'analizzare codice (dello stesso linguaggio)
  - tutti gli strumenti che manipolano il codice sono scritti essi stessi in Java: sono programmi Java che hanno bisogno di manipolare altri programmi Java
  - Esempi: compilatore, IDE, JUnit

La classe `java.lang.Class`

- Per ogni classe e per ogni interface Java esiste un oggetto che descrive il contenuto del file **`.class`** contenente il codice della classe
- Tale oggetto è istanza della classe **`Class`** (del package **`java.lang`**)
  - Attraverso i metodi offerti da **`Class`** è possibile analizzare (anche dinamicamente) tutte le caratteristiche della classe
    - è possibile ottenere l'elenco dei metodi, l'elenco dei supertipi, etc.
    - Vedi documentazione **`java.lang.Class`**

Ottenere il riferimento all'oggetto `Class`

- Tutte le classi hanno una variabile pubblica e statica chiamata **`class`** che memorizza un riferimento all'oggetto di tipo **`java.lang.Class`**
- Esempio:

```
Class classeBorsa =
    it.uniroma3.diadia.giocatore.Borsa.class;
```

Iterrogare un oggetto Class

- Possiamo interrogare l'oggetto Class per ottenere le proprietà della classe
- Esempio: otteniamo (e stampiamo) l'elenco dei metodi di una classe

```
import java.lang.reflect.Method;

public class EsempioRiflessione {
  public static void main(String[] args){
   Class classeBorsa =
         it.uniroma3.it.diadia.giocatore.Borsa.class;
   for (Method m : classeBorsa.getMethods())
         System.out.println(m);
  }
}
```

Uso della riflessione

- La riflessione si applica nello sviluppo di programmi avanzati con funzionalità complesse (ad esempio un IDE)
- Tuttavia, la riflessione offre una funzionalità particolarmente utile in molti contesti:
  - la possibilità di creare oggetti a partire dal nome della classe

Creazione di oggetti tramite la riflessione

- Fino ad ora abbiamo visto che la creazione di oggetti può avvenire solo tramite il costruttore
- Esempio:

```
Borsa borsa = new it.uniroma3.diadia.giocatore.Borsa();
```

- Con la riflessione è possibile creare oggetti invocando il metodo **`Object newInstance()`** di **`Class`**
- Esempio:

```
Class classeBorsa = it.uniroma3.diadia.giocatore.Borsa.class;
Borsa borsa = (Borsa)classeBorsa.newIstance();
```

Creazione di oggetti tramite la riflessione

- Fino a qui niente di speciale …
- Ma consideriamo un'altra possibilità che ci offre la classe **`Class`**
  - Attraverso il metodo statico **`Class forName()`** di possiamo cercare ed eventualmente caricare l'oggetto **`Class`** di una classe, dato il suo nome sotto forma di stringa
- Esempio:

```
Class classeBorsa =
    Class.forName("it.uniroma3.diadia.giocatore.Borsa");
```

Creazione di oggetti tramite la riflessione

- Usando questi metodi possiamo creare oggetti a partire dal nome di una classe
- Esempio:

```
Class classeBorsa =
     Class.forName("it.uniroma3.diadia.giocatore.Borsa");
Borsa borsa = (Borsa)classeBorsa.newIstance();
```

Applicazioni

- Possiamo scrivere codice in cui vengono creati dinamicamente oggetti a partire dal nome della loro classe
- Un esempio nel nostro studio di caso: possiamo creare gli oggetti Comando a partire dal nome del comando

Riflessione nello studio di caso

- Scriviamo una classe che implementa FabbricaDiComandi sfruttando la riflessione: riusciamo in maniera semplice ed elegante a eliminare la fastidiosa fisarmonica!
- L'idea è quella di costruire il nome della classe che processa un comando a partire dal nome del comando. Es. vai -> ComandoVai
- Carichiamo la classe per nome, istanziamo un oggetto

FabbricaDiComandiRiflessiva

```
public class FabbricaDiComandiRiflessiva implements FabbricaDiComandi {

    public Comando costruisciComando(String istruzione) {
        Scanner scannerDiParole = new Scanner(istruzione);
        String nomeComando = null;
        String parametro = null;
        Comando comando = null;

        if (scannerDiParole.hasNext())
                nomeComando = scannerDiParole.next(); // prima parola: nome del comando
        if (scannerDiParole.hasNext())
                parametro = scannerDiParole.next();// seconda parola: eventuale parametro
        try {
                String nomeClasse = "it.uniroma3.diadia.comandi.Comando";
                nomeClasse += Character.toUpperCase(nomeComando.charAt(0));
                nomeClasse += nomeComando.substring(1);
                comando = (Comando)Class.forName(nomeClasse).newInstance();
                comando.setParametro(parametro);
        } catch (Exception e) {
                comando = new ComandoNonValido();
                comando.setParametro("Comando inesistente");
        }
        return comando;
    }
}
```

Osservazioni

- Il metodo **`newInstance()`** invoca il costruttore no-arg: questo deve essere accessibile
  - E' per questo motivo che nella interface **`Comando`** abbiamo introdotto il metodo **`setArgomento(String)`**: sarebbe stato meglio passare l'argomento attraverso un costruttore, ma questa scelta ci avrebbe privato della possibilità appena descritta
- Esistono approcci per invocare costruttori con parametri, ma sono particolarmente complessi e approfondire oltre la riflessione va oltre gli obiettivi del corso