Java: Additional Features

Sommario: 16 Maggio, 2019

- ADT: Non Esposizione della Rappresentazione
- Equivalenza di Tipi e Assegnamento
- Uguaglianza di valori: ==, equals
- Duplicazione di valori: clone
- Presentazione di valori: toString
- ADT per valori strutturati: elements.
- Collection: Enhanced for (o for each)

ADT: Ancora una condizione

- ADT emulati in Java mediante classi e modificatori
- 3 condizioni:
  - **Stato Privato**
    Implementazione dei valori Inaccessibile
  - **Segnatura Pubblica**
    Uniche operazioni usabili dall'esterno della classe
  - **Esposizione Stato**
    Parametri trasmessi e Valori Calcolati delle operazioni pubbliche non devono mostrare parti dello stato.
    Esempio. `WrongStackImm2ADT` nell'allegato `stack`

Definition (Condizione di Non Esposizione dello Stato)

La stato della rappresentazione concreta non deve essere esposto in nessuna parte nè attraverso parametri nè attraverso il valore calcolato di un metodo pubblico. Quando la condizione è soddisfatta, l'ipotesi induttiva I(c) può essere assunta su c prima della invocazione di un metodo se provata vera sui soli costruttori.

ADT: Esposizione dello stato

Definition (Condizione di Non Esposizione dello Stato)

La stato della rappresentazione concreta non deve essere esposto in nessuna parte nè attraverso parametri nè attraverso il valore calcolato di un metodo pubblico. Quando la condizione è soddisfatta, l'ipotesi induttiva I(c) può essere assunta su c prima della invocazione di un metodo se provata vera sui soli costruttori.

```
interface APISTKImm{
    public APISTKImm push(int k);
    public int top();
    public APISTKImm pop();
    public boolean isEmpty();
    public Vector<Integer> show();
        // added for content inspection
}
public class WrongStackImm2ADT implements APISTKImm{
    private Vector<Integer> p;
    //AF and I
    public WrongStackImm2ADT(){...}
    public WrongStackImm2ADT push(int k){...}
    public int top(){...}
    public WrongStackImm2ADT pop(){...}
    public boolean isEmpty(){...}
    public Vector<Integer> show(){
        return p;
    }
}
```

Equivalenza di Tipi e Assegnamento

Definition (Strongly Typed, ST, Programming Language)

Ad ogni costruzione (espressione o comando) c, di ogni programma (legale) possiamo associare (a compile time) un **tipo unico** (un SuperTipo del tipo effettivo):

$$(\forall\, \texttt{c})(\exists!\, \texttt{T})\ \texttt{c}:\texttt{T}$$

- Java è un Linguaggio ST ("fortemente tipato"):
- I Tipi di un Linguaggio ST hanno relazioni di equivalenza:
  - Strutturale e/o;
  - Nominale
- I Tipi di Java hanno relazione di equivalenza Nominale:

  `T1 equivalente T2 sse T1 è T2`
- Assegnamento:

  $(\texttt{x} = \texttt{e}) : \texttt{T1}$ `sse` $(\texttt{x:T1} \wedge \texttt{e:T2} \wedge \texttt{T1>T2})$

Equivalenza di valori: ==, equals

- Categorie di Oggetti in Java: Due Categorie in accordo al **comportamento atteso**
  - **Modificabili (Mutable)**
    - `Stato dell'oggetto può cambiare`
  - **NonModificabili (Immutable)**
    - `Stato dell'oggetto non può cambiare`
- Per l'equivalenza di valori Java possiede:
  - operatore ==
    - `v1 == v2 sse stesso` **reference** `in memoria`
    - Corretto solo per Mutable e valori scalari (int, char, ...)

Metodo equals ed Overriding

- Metodo **equals**
  - Definito in Object ed ereditato da tutte le classi

    ```
    public boolean equals(Object o)
    ```
  - Su oggetti calcola come "==";

- **Mutable**
  - Uguali solo se sono lo stesso oggetto
  - equals **ereditata** calcola già correttamente
  - Overriding: NO, usare l'ereditata

- **Immutable**
  - Uguali se tutti i field corrispondenti sono uguali
  - equals ereditata non è adatta
  - Overriding: Si, per controllare i valori dei field corrispondenti

ImPairADT: Overriding di equals

- Come si esprime un overriding di **equals** quando usiamo poli- morfismo generico?
- **equals** ha segnatura con polimorfismo Object
  `public boolean equals(Object o)`
- Usare cast da Tipo Object a Tipi generici ma a variabili anonime: "?"
- Non è necessario controllare che `"(ImPairADTX<?,?>)o"` ab- bia proprio `A` e `B` come variabili generiche.

```
public class ImPairADTX <A,B> implements Cloneable{
    private final A left;//da nascondere
    private final B right;//da nascondere
    public ImPairADTX (A x, B y) {
        left = x;
        right = y;
    }
    public A getLeft(){...}
    public B getRight(){...}
    public boolean equals(Object o){//override equals
        ImPairADTX<?,?> ok;
        try{ok = (ImPairADTX<?,?>)o;}
        catch(Exception e){return false;}
        return (left.equals(ok.left) && right.equals(ok.right));
    }
```

Duplicazione di Oggetti: "shallow", "deep", o "mista"?

- **Scopo**: Creare una copia `o.clone()` distinta di un oggetto `o`.

- **Copia distinta** significa soddisfare 3 proprietà:
  1) Identica Classe: `o.getClass() = o.clone().getClass()`[1]
  2) Oggetti Distinti: `o != o.clone()`
  3) Oggetti Uguali (in comportamento): `o.equals(o.clone())`

- **Soluzioni Diverse** soddisfano le 3 proprietà: Sia `o` un oggetto della classe `A`.
  - **Copia Shallow di** `o`. È un oggetto di `A` i cui campi contengono i valori dei corrispondenti campi di `o`.
  - **Copia Deep di** `o`. È un oggetto di `A` i cui campi contengono una Copia Deep dei corrispondenti campi di `o`.
  - **Copia Intermediate di** `o`. ... i cui campi contengono una Copia Intermediate dei corrispondenti campi di `o`.

[1] `getClass()` metodo di Object fornisce identificativo unico della classe del target a cui è applicata.

Equivalenza Strutturale di Valori di Stesso Tipo

- **Equivalenza Strutturale di Valori.** È la forma di equivalenza dei valori utilizzata nella Programmazione Procedurale anche per valori modificabili: Due liste sono uguali se contengono la stessa disposizione di elementi.

Definition (Equivalenza Strutturale in Java)

Siano o1 e o2 oggetti di uno stesso tipo (i.e. classe) `T`. Sia, $\texttt{AF}_\texttt{T}$ la Funzione di Rappresentazione data per gli oggetti della classe T. Allora:

o1.equals(o2) **sse** $\texttt{AF}_\texttt{T}(\texttt{o1}) = \texttt{AF}_\texttt{T}(\texttt{o2})$.

- **Mutable e Immutable:** Con Equivalenza Strutturale dei Valori.
  - Uguali se tutti i field corrispondenti sono uguali
  - equals ereditata non è adatta
  - Overriding: Si. Controllare che i valori dei field corrispondenti siano equals.
- **Vector** e varie altre (ma non tutte) classi di Java usano Equivalenza Strutturale.
- **clone e equals** Quando equals controlla equivalenza strutturale, allora anche la proprietà (3) della definizione di clone può essere soddisfatta..