Programmazione II

7. Funzione di Astrazione e Invariante di Rappresentazione (PDJ 5.5-)

https://www.baeldung.com/java-record-keyword

Dragan Ahmetovic

Strumenti per comprendere la definizione dei tipi di dato

Processo di sviluppo: dall'Astrazione all'Implementazione

Il passaggio dall'**Astrazione** all'**Implementazione** si compone di tre fasi:

- **Specifica**: definizione dell'**Astrazione** e l'identificazione dei suoi **comportamenti**
  - Definire in maniera astratta cos'è il **dato** e che **operazioni** può svolgere
- **Selezione degli attributi**, ovvero l'**implementazione** della **Rappresentazione** del tipo di dato
  - Definire la **combinazione** di attributi che **modella correttamente** il nuovo tipo
- **Implementazione** delle procedure, ovvero l'**approccio specifico** per risolvere il problema
  - Definire il codice che correttamente implementa le operazioni specificate

Strumenti per comprendere la definizione dei tipi di dato

Legame tra Astrazione e Implementazione

Per comprendere il legame tra **Astrazione** e **Implementazione**, Liskov introduce 2 concetti:

- La **Funzione di Astrazione** (**Abstraction Function**)
  - Lega il **tipo di dato astratto che voglio rappresentare**, con il **tipo di dato concreto che uso**.
- L'**Invariante di Rappresentazione** (**Representation Invariant**)
  - Valuta la **correttezza** della **rappresentazione** in base all'astrazione data

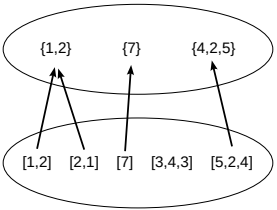
Questi concetti sono considerati su **una Astrazione** ed **una corrispondente Implementazione**.

Possono (e saranno) implementati come metodi dei tipi considerati

Funzione di Astrazione

Dall'astrazione all'implementazione

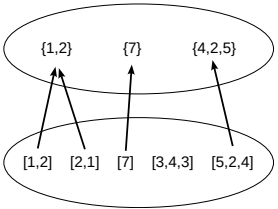
- Una data astrazione $\mathcal{A}$ definisce una serie di **oggetti astratti** $a \in \mathcal{A}$
- La classe $\mathcal{C}$ implementa l'astrazione $\mathcal{A}$ associando $\forall a$ la sua **rappresentazione** $c \in \mathcal{C}$
- La **rappresentazione** è uno specifico stato degli **attributi** $c = \{c_1, c_2, ...c_n\}$ dell'oggetto
- La **Funzione di Astrazione** associa alla rappresentazione c la corrispondente astrazione a
  - $AF : \mathcal{C} \rightarrow \mathcal{A}$
  - **Dominio**: insieme di **oggetti validi** $c \in \mathcal{C}$, ovvero tutte le combinazioni valide di stato degli attributi di c
  - **Codominio**: tutti gli oggetti astratti $a \in \mathcal{A}$
- Esempio:
  - astrazione: insieme di interi (senza ripetizioni)
  - rappresentazione: array di int



Funzione di Astrazione

Proprietà della Funzione di Astrazione

- **Suriettiva**, ovvero **ogni oggetto astratto** deve essere mappato a una **rappresentazione valida**
- **Non-iniettiva**, ovvero possono esserci **più rappresentazioni valide** per un oggetto astratto
  - quale rappresentazione sarà usata dipende dalla **implementazione**
- Possono esistere **rappresentazioni non valide**, che non corrispondono ad un oggetto astratto.
  - queste chiaramente **non fanno parte del dominio** della funzione
  - se la rappresentazione assume uno di questi stati, diventa **incoerente**
  - non modella più un'astrazione valida -> **grave problema**



Invariante di Rappresentazione

Identificazione delle rappresentazioni valide

Bisogna trovare un modo per **verificare** che una data **rappresentazione** sia **corretta**

- L'**Invariante di Rappresentazione** è una funzione $RI : \mathcal{C} \rightarrow \{True, False\}$
  - $RI(c) =$ '*True*' $\implies \exists a \in \mathcal{A} | AF(c) = a$
  - Se c è una **rappresentazione valida** di un oggetto astratto a, allora $RI(c)$ è vera
  - **Dominio**: tutti i possibili stati di c
  - **Codominio**: *True*, *False* (è un **predicato**)
- Nota: l'insieme degli elementi $c \in \mathcal{C} | RI(c) =$ '*True*' è il **dominio della Funzione di Astrazione**

Quindi quando una rappresentazione è valida?

- Se valgono **vincoli** specifici **su attributi** (specificati come congiunzione di predicati)
  - Es: studente deve avere un nome valido **E** una matricola valida
- Ci possono essere delle **combinazioni** di attributi valide o non valide o controlli più complessi
  - Es: un gatto maschio non può avere il pelo di tre colori diversi

Implementazione di AF e RI

Implementazione della Funzione di Astrazione

La Funzione di Astrazione dovrebbe restituire l'**oggetto astratto** rappresentato da *c*

- Non possiamo restituire l'oggetto astratto ma possiamo dare una sua **descrizione univoca**
  - È il comportamento desiderato di `` `toString()` ``
  - Nota: non restituire il valore di tutti gli attributi, ma la descrizione dell'astrazione rappresentata

Implementazione della Invariante di Rappresentazione

Per la Invariante di Rappresentazione Liskov suggerisce di implementare un nuovo metodo

- `` `public boolean repOk()` ``
  - Il dominio della funzione sono tutte le possibili combinazioni degli attributi
  - Il codominio è un valore booleano
- È possibile implementarla in Java!
- Posso **evitare rappresentazioni errate** in seguito alle **mutazioni** lanciando un'eccezione

Implementazione di AF e RI

Validità della Invariante di Rappresentazione

L'Invariante di Rappresentazione è un **vincolo interno all'implementazione**

- Garantisce che la rappresentazione sia valida **al termine** di ciascuna mutazione
  - Non è però garantita la validità **durante** una mutazione
  - Es: posso aggiungere un duplicato agli esami dello studente e poi levarlo
- In ogni caso non garantita se si **espone la rappresentazione** (modifiche esterne agli attributi)
  - Es: accesso esterno all'ArrayList degli esami può modificarla in maniera che non rispetta la RI
- Deve essere **corretta** e **completa**
  - Deve essere limitata a controlli necessari e validi (non controllare vincoli non richiesti)
  - Se mi dimentico qualche vincolo, il controllo è parziale (controllare tutti i vincoli richiesti)

Correttezza dei Tipi di Dati

Preservazione della Invariante di Rappresentazione

- Per un tipo di dato è fondamentale garantirne la correttezza
  - come **Dimostrare** che la **rappresentazione** sia sempre **corretta**?
  - Non si tratta di verificare che una certa sequenza di istruzioni sia corretta
  - Bisogna verificare la correttezza **dopo le possibili modifiche apportate** dai metodi
  - Possono avvenire in **qualunque ordine*** e con **qualunque valore dei parametri**
- Si procede per **induzione** (Induzione su tipo di dato):
  - Si verifica che il **costruttore** generi **sempre** un oggetto con RI valida
  - Si verifica che, dato un oggetto valido, **ogni metodo mutazionale** preservi la RI valida
  - Si verifica che i **metodi di produzione** generino oggetti validi
  - È necessario considerare l'influenza dei parametri dei metodi
  - Non è necessario verificare i metodi che osservano e basta
- Questa `sanity check` è da fare quando si termina l'implementazione

Correttezza dei Tipi di Dati

Invariante di Astrazione (Abstraction Invariant)

Un'altro concetto che torna utile è l'**Invariante di Astrazione**

- Mentre l'Invariante di Rappresentazione verifica la validità di una possibile rappresentazione
- L'Invariante di Astrazione aiuta a verificare la validità dell'astrazione, ovvero la **specifica**
- Per ogni **specifica procedurale** verificare se l'**astrazione è mantenuta integra**
  - cioè se la specifica rispetta l'astrazione del comportamento che il dato dovrebbe avere
- Procede per **induzione** e nella stessa maniera della RI
  - dal costruttore ai metodi mutazionali e di produzione
  - anche qua gli observers possono essere ignorati
- Da fare quando si termina la specifica
- Esempi di controllo di AI:
  - Un insieme di interi può avere elementi interi non ripetuti
  - La dimensione di un insieme di interi è pari al suo numero di elementi interi non ripetuti

Correttezza dei Tipi di Dati

Correttezza delle operazioni

A differenza di prima non voglio verificare la coerenza dell'astrazione o della rappresentazione del tipo di dato ma la **correttezza dell'implementazione**

- Si verifica che l'**implementazione soddisfi l'astrazione** usando la AF
  - Da verificare **non solo su metodi mutazionali** (e di produzione) ma anche **osservazionali**
  - Sui metodi osservazionali **verifico che l'osservazione corrisponda all'astrazione**
- Anche qua bisogna considerare l'effetto dei **parametri**
- Anche qua si procede per **induzione**
  - Assumendo che la RI sia valida prima di un metodo
  - Assumendo che lo sia anche al termine
  - Si parte dal costruttore, si verificano i metodi con strategia bottom-up
- Da verificare al termine dell'implementazione

Asserzioni

Limiti di `repOk()`

La Invariante di Rappresentazione, implementata come metodo, può avere delle limitazioni

- Verificare tutti i vincoli può essere **computazionalmente costoso**
- Spesso una **mutazione** coinvolge uno o **pochi attributi**, non serve controllare tutto
- Voglio controllare la repOK solo in fase di debug

Asserzioni

- (Relativamente nuovo) meccanismo di Java che permette di mettere controlli per debug
- Sintassi: `'assert e'` dove `'e'` è una espressione booleana
  - repOK() restituisce un boolean quindi può essere usata per assert
- ATTENZIONE: assert funzionano **solo se** programma **invocato** con `java -ea <Classe>`

Astrazioni e Rappresentazioni - Mutabili e Immutabili

Astrazioni (im)mutabili

Durante la definizione dell'astrazione è possibile stabilire che il tipo di dato sia:

- **Mutabile** - sono possibili modifiche al del dato
  - Es: posso aggiungere nuovi esami allo studente
- **Immutabile** - non sono possibili modifiche al tipo di dato
  - Es: non posso modificare un Punto creato, posso crearne di altri

Rappresentazioni (im)mutabili

Anche se un'astrazione è **immutabile**, la sua rappresentazione può essere **mutabile**

- Un'**astrazione immutabile** non deve avere metodi che possano impattare il risultato della **AF**
  - **Effetti collaterali benevoli** sono ancora possibili
- Per rendere la **rappresentazione immutabile** si usa la keyword **final** sugli attributi

Astrazioni e Rappresentazioni - Mutabili e Immutabili

Effetti collaterali benevoli

Poiché possono esserci più rappresentazioni valide per un oggetto astratto, è possibile fare modifiche alla rappresentazione che però non hanno effetto sulla AF

- Questi **effetti collaterali benevoli** possono essere utili per questioni di efficienza
- Utilizzo di algoritmi di ricerca più efficienti
  - es: Ordinare l'array che rappresenta l'insieme di interi
  - l'ordine degli elementi nell'array non influisce sull'astrazione
- Caching di valori per non doverli ricalcolare
  - es: una lista di valori crescenti e voglio sapere la frequenza del valore massimo
  - caching del conteggio del valore massimo per non dover scorrere la lista ogni volta
- possibile anche per **astrazioni immutabili** (chiaramente con la **rappresentazione mutabile**)

Record

Record

Per modellare tipi composti, senza comportamenti si possono creare classi **Record** senza metodi

- Utili per fare array o ArrayList di N-uple di elementi
  - (invece di dover fare molteplici array(List) allineati)
- Si possono fare a mano:
  - come Classi costituite da attributi + un costruttore per assegnare i valori iniziali
  - Gli attributi si possono accedere direttamente (attenzione a non esporli se sono mutabili!)
  - Non serve RI, AF è identitaria (riporta valori di tutti gli attributi)
  - potrebbe servire reimplementare `toString`, `equals`, `clone`, `hashCode`
- Da Java 14 esiste anche keyword `record` per la creazione automatica di record immutabili

Esempio record (new):

```
1 public record Person(String name, String address) {
2   public Person { //compact constructor opzionale, constructor classici possibili
3     Objects.requireNonNull(name); //se null lancia NullPointerException
4     Objects.requireNonNull(address);
5   }
6 }
```

Record

Esempio record (old):

```
1  public class Coord {
2  //OVERVIEW: record che modella coordinate sul piano cartesiano
3    double x, y;
4
5    public Coord(double x, double y) {
6      this x = x;
7      this y = y;
8    }
9
10   public boolean equals(Object o) {
11     if (!(o instanceof Coord))
12       return false;
13
14     if(this.x != (Coord)o.x || this.y != (Coord)o.y)
15       return false;
16
17     return true;
18   }
19
20   public int hashCode() {
21     return Objects.hash(this.x,this.y);
22   }
23
24   public Coord clone() {
25     try {
26       return (Coord) super.clone();
27     } catch (CloneNotSupportedException e) {
28       return null;
29     }
30   }
31
32   public String toString() {
33     return "Coord:[" + this.x + ", " + this.y + "]";
34   }
35 }
```

Programmazione II

7. Funzione di Astrazione e Invariante di Rappresentazione (PDJ 5.5-)

https://www.baeldung.com/java-record-keyword

Dragan Ahmetovic