AltriEsercizi7 - Esercizio1

Esercizio

Si consideri la classe ImPairADTe<A,B> per coppie IMMUTABLE, nell'omonimo file allegato in AltriEsercizi7. La classe contiene una definizione del metodo clone che vorrebbe creare una copia dello stato dell'oggetto assegnando ai vari campi un clone del valore contenuto nel corrispondente campo dell'oggetto. Purtroppo tale definizione non opera correttamente. In particolare:

(a) *Il compilatore segnala 3 errori, tutti sullo stm:* `new ImPairADTe<A,B>(left.clone(),right.clone())`, *fornendo la seguente motivazione principale: "...error: clone() has protected access in Object". Sulla base di quanto abbiamo visto a lezione sul metodo clone, si chiarisca il significato di tale motivazione.*

(b) *Sulla base del chiarimento in (a) si proponga una soluzione (motivandola e verificandola opportunamente).*

Soluzione

(a)

In effetti, il compilatore rileva un errore in 3 punti diversi dello stm. Il primo punto è nel termine `left.clone()` sul quale è emesso il messaggio "...error: clone() has protected access in Object". Limitiamoci a questo ed osserviamo che, in effetti, `left.clone()` vorrebbe invocare, ed applicare a left, il metodo clone di Object: Dopo tutto left è sicuramente anche un Object ed eredita tale metodo. Il metodo ha modificatore `protected`, come ricorda il messaggio di errore. Quindi left eredita un metodo clone() con modificatore `protected`. Possiamo invocare nella classe ImPairADTe<A,B> un metodo protected di una classe generica A? Si ma solo quando A è sottoclasse di ImPairADTe<A,B>. Purtroppo A non è dichiarata tale (e in Java8 non può esserlo).

(b)

Osservazione. Per classi di oggetti IMMUTABLE, come ImPairADTe<A,B>, non c'è alcuna ragione di richiedere un clone dei componenti dello stato, e il trattamento corretto per clonare gli oggetti della classe è:

- Dichiarare la classe sottoclasse di cloneable, definire un overridden **pubblico** della clone di Object
  vedi ImPairADTX<A,B> nel file ImPairADTX allegato

AltriEsercizi7 - Esercizio2

Esercizio

Si consideri la classe MuPairADTe<A,B> per coppie MUTABLE, nell'omonimo file allegato in AltriEsercizi7. La classe contiene una definizione del metodo clone che vorrebbe creare una copia dello stato dell'oggetto assegnando ai vari campi un clone del valore contenuto nel corrispondente campo dell'oggetto. Purtroppo anche tale definizione non opera correttamente. In particolare:

(a) *Il compilatore segnala 3 errori, tutti sullo stm:* `new MuPairADTe<A,B>(left.clone(),right.clone())`, *fornendo la seguente motivazione principale: "...error: clone() has protected access in Object". Sulla base di quanto abbiamo visto a lezione sul metodo clone, si chiarisca il significato di tale motivazione.*

(b) *Sulla base del chiarimento in (a) si proponga una soluzione (motivandola e verificandola opportunamente).*

Soluzione

(a)

Le ragioni sono esattamente le stesse esposte nella soluzione dell'ercizio precedente. Non possiamo invocare nella classe MuPairADTe<A,B> un metodo protected di una classe A a meno che A sia sottoclasse di MuPairADTe<A,B> o sia nello stesso package. Trattandosi di un generico A non siamo in nessuno dei 2 casi.

(b)

Per classi di oggetti MUTABLE dobbiamo imporre che le classi dei componenti modificabili dello stato siano anch'essi equipaggiati di un operazione clone per valori MUTABLE: ovvero devono avere una clone con modificatore PUBLIC e definizione che deve essere specifica di ogni classe MUTABLE. Quindi dobbiamo usare un meccanismo che imponga alle classi MUTABLE e con operazione clone una clone con segnatura PUBLIC. Questo conduce a:

1) Introdurre un'interfaccia MuCloneable con un'operazione clone pubblica che estenda l'interfaccia Cloneable;
2) Richiedere che i tipi dei componenti modificabili siano tutti sottotipo di MuCloneable;
3) Definire nella classe clone ricorrendo ai cone dei componenti (come, peraltro, già fatto nello stm "incriminato")
   vedi MuPairADTX<A,B> nel file MuPairADTXXXX allegato

AltriEsercizi7 - Esercizio3

Esercizio

La soluzione proposta nell'esercizio2 per quanto operi "correttamente" appare piuttosto estrema. Infatti, prevede che tutti i componenti dello stato siano MUTABLE e duplicati come tali. Questa soluzione diventa inutilmente dispendiosa quando abbiamo anche componenti IMMUTABLE come ad esempio, un `Integer`*.*

(a) *Si consideri il caso di uso introdotto in ImPairADTX.java allegato, che introduce coppie IMMUTABLE a componenti* `Integer` *e* `String`*, rispettivamente. E lo si tratti nel contesto della classe MuPairADTX.java per coppie MUTABLE, definita nel file MuPairADTXXXX allegato.*

(b) *Formulare una soluzione che sia in grado di distinguere le caratteristiche (MUTABLE/IMMUTABLE) dei componenti dello stato dell'oggetto da clonare. Allo scopo si usi l'operatore relazionale* `instanceof`*: È un operatore binario con la seguente forma sintattica:*

```
object instanceof class
```

dove, gli argomenti devono essere un oggetto e un tipo, rispettivamente. L'espressione calcola `true` *se* `object` *appartiene ad una sottoclasse della classe (o, interfaccia)* `class`*.*

Soluzione

(a)

In questo caso entrambi i componenti sono IMMUTABLE e non richiedono un duplicato, ovvero un'applicazione di clone(). Ma l'uso di MuPairADTXXXX.java non permette di distinguere e ci obbliga a trattarli comeMUTABLE e quindi duplicarli (con consumo di tempo e spazio del tutto inutili). Per farlo occorre notare che nè Integer nè String hanno un metodo clone() con modificatore public, quindi non possono essere direttamente utilizzati nello stato di un oggetto MUTABLE che li voglia duplicare. Dobbiamo ancora una volta ricorrere all'uso di tipi `ref<T>` che possono racchiudere un qualunque T e sono opportunamente equipaggiati dei metodi addizionali, incluso un metodo public clone().

vedi MuPairADTX<A,B> e le classi ref<T> e main, nel file MuPairADTXXXX.java allegato

(b)

Otteniamo una soluzione MuPairADTX<A,B> che non impone vincoli sui tipi `A` e `B`. Definisce un metodo `public clone()` che controlla se `A` (risp. `B`) sia sottoclasse di MuCloeneable e in caso affermativo, provvede ad applicare il corrispondente metodo clone() della classe `A` (risp. `B`).

vedi MuPairADTX<A,B> nel file MuPairADTX.java allegato

AltriEsercizi7 - Esercizio4

Esercizio

Si consideri la classe PolyJADT, nell'omonimo file allegato, per polinomi MODIFICABILI.
(a) Si completi la classe con gli opportuni additionals e
(b) Si estenda il caso di uso al loro impiego.

Soluzione