

Sommario: 22 Maggio, 2018

- **Complementamenti:**
  - > Metodo `elements` e `Iteratori` (Lezione17)
- **Il Tipo di Dato `Tree` in Java**
  - > Stato: Struttura `MultiComponente`: `AF & I`
  - > Interfaccia `Label` e `Polimorfismo Object`
  - > Costruttori: Scelta
  - > Eccezioni e Operazioni: `getLabel`, `getSons`, `add`, `remove`
  - > Gli `Additional`: `equals`, `toString`, `clone`
  - > `clone`: Valori dei Componenti e Struttura dello Stato
  - > `clone`: Copia `Shallow`, `Deep` o `Intermediate`
- **Una Versione con Polimorfismo Generico: `Tree<T>`**
- **Il Tipo Astratto con Polimorfismo Generico: `API<T>`**
  - > Usiamo `Tree<T>` e scriviamo una `API<T>` adatta
  - > É adatta? Verifichiamolo con il compilatore
  - > Problemi: Perché?
  - > Soluzioni: Perché?
- **Esercizi**

- metodo **`elements`**:

- Da definire in classi di oggetti strutturati: Liste, Alberi, Code, Insiemi...
- Fornisce un valore `Collection` dei valori contenuti nel valore (astratto):
  - tutti gli elementi della lista
  - tutti i nodi dell'albero (oppure, tutti gli archi)
  - tutti gli oggetti nella coda
  - ...
- Lo esprimeremo con:

```
public LinkedList<T> elements()
```

## Collezione degli elementi: elements /2

```
public LinkedList<A> elements(){
    //lista Valori Label
    if (label == null) return new LinkedList<A>();
    LinkedList<A> ret = sons.elements();
    ret.add(label);
    return ret;
}
```

Tipo Astratto API<A> in TREES/MuGAETREE/Tree.java

- **v.elements()**: Fornisce una Collection dei valori contenuti nel valore strutturato v
  - Permette di aumentare l'**usabilità** dei Tipi Astratti proteggendone l'**integrità**
  - Migliora l'**usabilità** dei Tipi di Dato indicando un'operazione preposta allo scopo

### Definition (Integrità di Valore o Dato)

Indica l'assenza di alterazioni non previste durante l'intera vita del valore.

- Interfaccia Collection è per valori che esprimono:
  - Collezioni di valori
  - È superTipo di classi importanti tra cui:  
`Vector<T>`, `LinkedList<T>`
- Utilizzabili nell'iterazione mediante *enhanced for*:
  - Sia `Coll<T>` una collection di valori di tipo `T`.
  - Sia `C` un oggetto di tipo `Coll<T>`.
  - Sia `code(x)` un codice nella variabile (libera) `x` di tipo `T`  
`for(T x: C) code(x)`
  - Itera `code(x)` su ogni valore `u` di tipo `T` che sia in `C`;
  - Ad ogni iterazione `x` è legato ad un diverso `u` in `C`;
  - Ordine dei legami è ignoto e deve essere inessenziale per il programma
- Enhance for è analogo a `List.foldLeft`, in OCaml, dove la collection è la lista e `code(x)` è la funzione da iterare.

# elements e enhanced for

```
class Test6{//elements
    public static void main(String[] args) throws CloneNotSupportedException{
        Tree<String> t1 = new Tree<String>("A",null);
        Tree<String> t2 = new Tree<String>("B",null);
        t1.add(1,t2);
        Tree<String> t8 = t1.clone();
        Scanner StdIn = new Scanner(System.in);
        String search = StdIn.next();
        System.out.println("le etichette dei nodi di " + t8 + " sono " + t8.elements());
        for(String x: t8.elements()){
            if (x.equals(search)){
                System.out.println(search + " etichetta un nodo di " + t8);
                return;
            }
        }
        System.out.println(search + " non etichetta alcun nodo di " + t8);
    }
}
```

Tipo Astratto API<A> in TREES/MuGAETREE/Test.java

- **Stato Concreto:AF & I**

```
public class Tree implements Cloneable {
    private Label label;
    private Vector<Tree> sons;
    /*
    AF(c) = < > iff c.label==null
    AF(c) = <a - t1 ... tn> iff (c.label == a)&&(c.sons.size() == n)&&
                               (forall i\in[1..n], c.sons.get(i-1) == ti)
    I(c) = (c.label == null)|| (c.label != null && c.sons != null)
    */
    ...
}
```

- > La forma  $\langle a - [t_1, \dots, t_n] \rangle$  della presentazione si desume dalla definizione di `toString()`, data nel testo.

# Interfaccia Label e Polimorfismo Object

- Polimorfismo Object: I Tree sono etichettati su Label che è un'Interfaccia.

```
public interface Label extends Cloneable{
}
```

- > Riutilizzo del codice Tree su ogni classe A che "implementi" Label

```
class A implements Label{String v; ...}
class B implements Label{Integer v;...}
...
class Z implements Label{...}

class Test6{//Polimorfismo Object
    public static void main(String[] args) {
        Tree t1 = new Tree(new B(100),null);
        Vector<Tree> sons1 = new Vector<Tree>();
        sons1.add(t1);
        Tree t2 = new Tree(new A("A"),sons1);
        System.out.println("t1 = " + t1);
        System.out.println("t2 = " + t2);
    }
}
```

- > Esecuzione:

```
etruria-wifi-217-207:MuObjTREE marcob$ java TreePack/Test6
t1 = <100 - >
t2 = <A - [<100 - >]>
etruria-wifi-217-207:MuObjTREE marcob$
```

- In questo corso non approfondiamo oltre questa forma di Polimorfismo interessante ma con proprietà non del tutto chiare.

- il Testo indicava la signature di 2 costruttori da definire:

```
public Tree(){  
}  
public Tree(Label label, Vector<Tree> sons){  
    ...  
}
```

- > Il primo crea alberi vuoti (e lo useremo per definire clone in forma appropriata allo stato utilizzato in Tree)
- > Il primo ha corpo `super()` di default

- Entrambi devono rispettare AF&I:

```
public Tree(){  
}  
public Tree(Label label, Vector<Tree> sons){  
    this.label = label;  
    this.sons = new Vector<Tree>();  
    if (sons == null) return;  
    for (int i=0; i<sons.size(); i++){//copiare il contenuto  
        this.sons.add(i,sons.get(i)); //per evitare esposizione  
        //della rappresentazione  
    }  
}
```

- Eccezioni: Potremmo anche non introdurre di nuove
  - NullPointerException, OutOfBoundsException, IllegalArgumentException: Sono tutte RuntimeException.
  - Sono tutte sollevate da Vector.add e Vector.remove, quando invocate da add e remove, definite nel testo.
- getLabel: Banale, restituisce il valore del componente label

```
public Label getLabel(){  
    return label;  
}
```

- getSons: Restituisce una copia della struttura con i valori

```
public Vector<Tree> getSons(){  
    Vector<Tree> ret = new Vector<Tree>();//copiare il contenuto  
    for (int i=0; i<sons.size(); i++){ //per evitare esposizione  
        ret.add(i,sons.get(i)); //rappresentazione  
    }  
    return ret;  
}
```

- La struttura Vector dei valori deve essere nuova
- I valori (Immutable o no) devono essere gli stessi contenuti nella struttura

- I Tree sono Valori MUTABLE
- `Object.equals()` è definita per calcolare correttamente con i MUTABLE
- `Object.equals()` è ereditata ed usabile se non ridefinita in `Tree`.
- ERRORE fornire in `Tree` una nuova definizione di `equals`.

- Deve essere sempre ridefinita per fornire come tipo calcolato la classe invece di Object
- Dobbiamo creare una copia dell'oggetto con value reference differente e stato:
  - Copia Shallow
  - Copia Deep
  - Copia Intermediate
- Diamo la definizione per ciascuno dei 3 e vediamone il comportamento su un test di uso.

- Diamo la definizione per ciascuno dei 3 e ne vediamo il comportamento sul seguente test di uso.

```
class Test5{//clone a Copia ...
    public static void main(String[] args) throws CloneNotSupportedException{
        Tree t1 = new Tree(new A("A"),null);
        Tree t2 = new Tree(new A("B"),null);
        t1.add(1,t2);
        Tree t8 = t1.clone();
        System.out.print("t1 = " + t1);
        System.out.println(", e t8 = " + t8);
        System.out.println("t1 e t8 stesso valore in questo istante,\nMa t1 uguale t8 in ogni
            possibile stato del calcolo? --- " + (t1.equals(t8)));
        //Il requisito su uguaglianza tra oggetto e suo clone solo per IMMUTABLE
        //-- Le etichette di un nodo non possono cambiare.
        // Domandiamoci se i due alberi hanno radice etichettata nello stesso modo
        System.out.println("t1 e il suo clone hanno stessa etichetta? --- " + (t1.getLabel().
            equals(t8.getLabel())));
        //-- I figli possono cambiare e vorremmo che cambiassero in modo indipendente
        // Domandiamoci se i due alberi sono indipendenti sui figli.
        t1.add(2,t8); //aggiunto 1 figlio a t8
        System.out.println("t1 ha figli: " + t1.getSons());
        System.out.println("t8 ha figli: " + t8.getSons());
    }
}
```

- Definiamo e proviamo su Test5 ciascuna delle 3 Copie.

# Additional: clone a Copia Shallow

```
public Tree clone()throws CloneNotSupportedException{
    return (Tree) super.clone(); //overriding a Copia Shallow
} //Meglio Copia Deep? Discutere
```

```
class Test5{//clone a Copia ...
    public static void main(String[] args) throws CloneNotSupportedException{
        Tree t1 = new Tree(new A("A"),null);
        Tree t2 = new Tree(new A("B"),null);
        t1.add(1,t2);
        Tree t8 = t1.clone();
        System.out.println("t1 = " + t1);
        System.out.println(", e t8 = " + t8);
        System.out.println("t1 e t8 stesso valore in questo istante,\nMa t1 uguale t8 in ogni
            possibile stato del calcolo? --- " + (t1.equals(t8)));
        //Il requisito su uguaglianza tra oggetto e suo clone solo per IMMUTABLE
        //-- Le etichette di un nodo non possono cambiare.
        // Domandiamoci se i due alberi hanno radice etichettata nello stesso modo
        System.out.println("t1 e il suo clone hanno stessa etichetta? --- " + (t1.getLabel().
            equals(t8.getLabel())));
        //-- I figli possono cambiare e vorremmo che cambiassero in modo indipendente
        // Domandiamoci se i due alberi sono indipendenti sui figli.
        t1.add(2,t8); //aggiunto 1 figlio a t8
        System.out.println("t1 ha figli: " + t1.getSons());
        System.out.println("t8 ha figli: " + t8.getSons());
    }
}
```

```
MarcoBelliasAir:MuTREE marcob$ javac Label.java Tree.java Test.java -d .
MarcoBelliasAir:MuTREE marcob$ java TreePack/Test5
t1 = <A - [<B - >>], e t8 = <A - [<B - >>]
t1 e t8 stesso valore in questo istante,
Ma t1 = t2 in ogni possibile stato del calcolo? --- false
t1 e il suo clone hanno stessa etichetta? --- true
Exception in thread "main" java.lang.StackOverflowError
...
```

# Additional: clone a Copia Deep

```
public Tree clone() throws CloneNotSupportedException{
    Tree ret = (Tree) super.clone(); //overriding a Copia Deep
    if (label instanceof Cloneable) ret.label = (Label)((A)label).clone();
    if (sons instanceof Cloneable) ret.sons = (Vector<Tree>)sons.clone();
    return ret;
} //Meglio Copia Shallow? (vedi WrongSol)

class Test5{//clone a Copia ...
    public static void main(String[] args) throws CloneNotSupportedException{
        Tree t1 = new Tree(new A("A"),null);
        Tree t2 = new Tree(new A("B"),null);
        t1.add(1,t2);
        Tree t8 = t1.clone();
        System.out.print("t1 = " + t1);
        System.out.println(", e t8 = " + t8);
        System.out.println("t1 e t8 stesso valore in questo istante,\nMa t1 uguale t8 in ogni
            possibile stato del calcolo? --- " + (t1.equals(t8)));
        //Il requisito su uguaglianza tra oggetto e suo clone solo per IMMUTABLE
        //Le etichette di un nodo non possono cambiare.
        // Domandiamoci se i due alberi hanno radice etichettata nello stesso modo
        System.out.println("t1 e il suo clone hanno stessa etichetta? --- " + (t1.getLabel().
            equals(t8.getLabel())));
        //-- I figli possono cambiare e vorremmo che cambiassero in modo indipendente
        // Domandiamoci se i due alberi sono indipendenti sui figli.
        t1.add(2,t8); //aggiunto 1 figlio a t8
        System.out.println("t1 ha figli: " + t1.getSons());
        System.out.println("t8 ha figli: " + t8.getSons());
    }
}
```

```
etruria-wifi-217-207:MuObjTREE marcob$ javac Label.java Tree.java Test.java -d .
```

```
Note: Tree.java uses unchecked or unsafe operations.
```

```
Note: Recompile with -Xlint:unchecked for details.
```

```
MarcoBelliasAir:MuTREE marcob$ java TreePack/Test5
```

```
t1 = <A - [<B - >>], e t8 = <A - [<B - >>
```

```
t1 e t8 stesso valore in questo istante,
```

```
Ma t1 = t2 in ogni possibile stato del calcolo? --- false
```

```
t1 e il suo clone hanno stessa etichetta? --- false
```

```
t1 ha figli: [<B - >, <A - [<B - >>]
```

```
t8 ha figli: [<B - >]
```

# Additional: clone a Copia Intermediate

```
public Tree clone()throws CloneNotSupportedException{
    Tree ret = (Tree) super.clone(); //overriding a Copia Intermediate
    if (sons instanceof Cloneable)//Clone solo sulla struttura della
        ret.sons = (Vector<Tree>)sons.clone(); //rappresentazione
    return ret;
} //Meglio Copia Shallow? o Copia Deep (vedi WrongSol)
class Test5{//clone a Copia ...
    public static void main(String[] args) throws CloneNotSupportedException{
        Tree t1 = new Tree(new A("A"),null);
        Tree t2 = new Tree(new A("B"),null);
        t1.add(1,t2);
        Tree t8 = t1.clone();
        System.out.print("t1 = " + t1);
        System.out.println(", e t8 = " + t8);
        System.out.println("t1 e t8 stesso valore in questo istante,\nMa t1 uguale t8 in ogni
            possibile stato del calcolo? --- " + (t1.equals(t8)));
        //Il requisito su uguaglianza tra oggetto e suo clone solo per IMMUTABLE
        //-- Le etichette di un nodo non possono cambiare.
        // Domandiamoci se i due alberi hanno radice etichettata nello stesso modo
        System.out.println("t1 e il suo clone hanno stessa etichetta? --- " + (t1.getLabel().
            equals(t8.getLabel())));
        //-- I figli possono cambiare e vorremmo che cambiassero in modo indipendente
        // Domandiamoci se i due alberi sono indipendenti sui figli.
        t1.add(2,t8); //aggiunto 1 figlio a t8
        System.out.println("t1 ha figli: " + t1.getSons());
        System.out.println("t8 ha figli: " + t8.getSons());
    }
}
```

etruria-wifi-217-207:MuObjTREE marcob\$ javac Label.java Tree.java Test.java -d .

Note: Tree.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

etruria-wifi-217-207:MuObjTREE marcob\$ java TreePack/Test5

t1 = <A - [<B - >]>, e t8 = <A - [<B - >]>

t1 e t8 stesso valore in questo istante,

Ma t1 uguale t8 in ogni possibile stato del calcolo? --- false

t1 e il suo clone hanno stessa etichetta? --- true

t1 ha figli: [<B - >, <A - [<B - >]>]

t8 ha figli: [<B - >]