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"` abbia 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.

Substitution Equivalence in Equals

- **Equivalenza per Sostituzione** È la forma di equivalenza dei valori strutturati utilizzata nella Programmazione Funzionale e in molte Applicazioni di Programmazione OO: Due valori sono uguali se l'uno può essere sostituito all'altro, nel programma in cui i due valori sono introdotti, senza che il programma risultante cambi comportamento.

Definition (Substitution Equivalence in Java (debole))

Sia `P` un programma di Java. Siano `o1` e `o2` oggetti di uno stesso tipo (i.e. classe) `T`, introdotti in `P`. Si indichi con `Q[o←o']`, la sostituzione di ogni occorrenza dell'oggetto `o` con l'oggetto `o'`, entrambi introdotti in un programma Java `Q`. Si indichi con `Sem Q` il comportamento del programma `Q`. Allora:

`o1.equals(o2)` **solo se** `Sem(P[o1←o2]) = Sem P = Sem (P[o2←o1])`.

- **Definita Sopra:** È quella con cui abbiamo dato la prima definizione di `equals`.
- **Mutable:** Tratta i Mutable come uguali solo a loro stessi.
- **Falsifica:** La 3ª proprietà richiesta dal metodo **clone**, vista sopra.

Structural Equivalence in Equals

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

Definition (Equivalenza Strutturale in Java (debole))

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) **solo se** $\texttt{AF}_\texttt{T}(\texttt{o1}) = \texttt{AF}_\texttt{T}(\texttt{o2})$.

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

Structural vs. Substitution Equivalence in Equals

- **Mutable.** Nessuna differenza. In entrambi i casi: Stesso Controllo dei Componenti, quindi stesso codice.
- **Immutable.** La structural equivalence richiede Stesso Controllo dei Componenti, richiesto per i mutable. Quindi stesso codice di `equals` per valori Mutable e per valori Immutable con equivalenza strutturale.
- **Esempio:** La definizione di `equals` data sopra per i valori Immutable di tipo `ImPairADTX<A,B>`, può essere utilizzata per definire la `equals` dei corrispondenti valori Mutable con tipo `MuPairADTX<A,B>`.

Duplicazione di Oggetti: clone() e il suo Meccanismo

Introduzione ed uso di un metodo **clone**: Poggia su un Meccanismo articolato in 5 parti.

- Definizione di un metodo **clone()** in **Object** con segnatura:
  `protected Object clone() throws CloneNotSupportedException`
- Tutte le classi ereditano **clone()** da **Object**;
- Esistenza di interfaccia **Cloneable**;
- Uso e Overriding di **clone()** ammesso solo per le classi che implementano **Cloneable**;
- **Object** non implementa **Cloneable**.

Comportamento di clone() definita in Object

- **clone()**
  - Signature:
    > `protected Object clone() throws CloneNotSupportedException`
  - Definizione e Comportamento:
    > `Definita in Object ed ereditata da tutte le classi;`
    > `Comportamento di o.clone():`
    dove: `o` di classe `A` e `clone()` metodo ereditato da Object
    - **genera** `error: clone() has protected access in Object`
      se invocato da classe fuori dalla protected area di `A`
    - **solleva** `CloneNotSupportedException`
      se invocato in protected area di `A` ma `A` non è `Cloneable`
    - **crea una Copia shallow** di `o`, altrimenti

- **Cloneable**. Tutte le classi in cui si voglia **clone()**

Duplicazione di Oggetti: IMMUTABLE

- **Overriding Obbligatorio** per rendere `clone`:
  - > public (usabile ovunque)
  - > calcolante duplicati "cast" sul Tipo della classe
  - > senza sollevare `CloneNotSupportedException`
  - > generante una copia shallow

- **Duplicato ottenibile** da `clone()` di Object:
  - > Invocando opportunamente, `super.clone()`

- Vediamo **Caso Generale** di Definizione ed Uso:
  - > Abbiamo una classe di oggetti a cui aggiungere il metodo clone()
  - > Abbiamo una seconda classe in cui i valori della prima devono essere duplicati

Duplicazione di Oggetti IMMUTABLE: Caso Generale

- Una classe di oggetti a cui aggiungere il metodo clone(): `ImPairADTe.java`

```
import java.lang.*;
import java.io.*;

class ImPairADTe<A,B> implements Cloneable{
    private final A left;//da nascondere
    private final B right;//da nascondere
    public ImPairADTe (A x, B y) {
        left = x; right = y;
    }
    public A getLeft(){//da aggiungere per uso effettivo
        return left;
    }
    public B getRight(){//da aggiungere per uso effettivo
        return right;
    }
    public boolean equals(Object o){//override equals
        ImPairADTe<?,?> ok;
        try{ok = (ImPairADTe<?,?>)o;}
        catch(Exception e){return false;}
        return (left.equals(ok.left) && right.equals(ok.right));
    }
    //clone: inserire definizione
    public String toString(){
        return ("("+left.toString()+","+right.toString()+")");
    }
}
```

- Una seconda classe che usa e duplica i valori della prima: `main`

```
class main{
    public static void main(String args[]){
        ImPairADTe <Integer,String> myPlayCard = new ImPairADTe<Integer,String>(3,"fiori");
        System.out.println("la carta è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        ImPairADTe <Integer,String> myPlayclone = myPlayCard.clone();
        System.out.println("Il clone è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        System.out.println("la carta e il suo clone sono uguali? :" + myPlayCard.equals(myPlayclone));
    }
}
```

Duplicazione di Oggetti IMMUTABLE: Proposta1

- Una definizione di clone(): È corretta?

```
class ImPairADTe<A,B> implements Cloneable{
    private final A left;//da nascondere
    private final B right;//da nascondere
    public ImPairADTe (A x, B y) {...}
    public A getLeft(){...}
    public B getRight(){...}
    public boolean equals(Object o){...}
    public ImPairADTe<A,B> clone(){
        return (ImPairADTe<A,B>) super.clone();
    }
    public String toString(){...}
}
```

- È corretta?: Compiliamo le due classi ed Eseguiamo (se possibile)

```
class main{
    public static void main(String args[]){
        ImPairADTe <Integer,String> myPlayCard = new ImPairADTe<Integer,String>(3,"fiori");
        System.out.println("la carta è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        ImPairADTe <Integer,String> myPlayclone = myPlayCard.clone();
        System.out.println("Il clone è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        System.out.println("la carta e il suo clone sono uguali? :" + myPlayCard.equals(myPlayclone));
    }
}
```

Duplicazione di IMMUTABLE. Proposta1: Errata

- Una definizione di clone(): È corretta?

```
class ImPairADTe<A,B> implements Cloneable{
    private final A left;//da nascondere
    private final B right;//da nascondere
    public ImPairADTe (A x, B y) {...}
    public A getLeft(){...}
    public B getRight(){...}
    public boolean equals(Object o){...}
    public ImPairADTe<A,B> clone(){
        return (ImPairADTe<A,B>) super.clone();
    }
    public String toString(){...}
}
```

- Compiliamo le due classi ed Eseguiamo (se possibile):

```
host-131-114-217-78:AltriEsercizi7Listings marcob$ javac ImPairADTeError.java -d .
ImPairADTeError.java:47: error: unreported exception CloneNotSupportedException; must be
caught or declared to be thrown
        return (ImPairADTe<A,B>) super.clone();
                                             ^
Note: ImPairADTeError.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
host-131-114-217-78:AltriEsercizi7Listings marcob$
```

- Correggiamo

Duplicazione di Oggetti IMMUTABLE: Proposta2

- Una definizione di clone(): È corretta?

```
class ImPairADTe<A,B> implements Cloneable{
    private final A left;//da nascondere
    private final B right;//da nascondere
    public ImPairADTe (A x, B y) {...}
    public A getLeft(){...}
    public B getRight(){...}
    public boolean equals(Object o){...}
    public ImPairADTe<A,B> clone(){
        try{return (ImPairADTe<A,B>) super.clone();}
        catch (CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

- È corretta?: Compiliamo le due classi ed Eseguiamo (se possibile)

```
class main{
    public static void main(String args[]){
        ImPairADTe <Integer,String> myPlayCard = new ImPairADTe<Integer,String>(3,"fiori");
        System.out.println("la carta è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        ImPairADTe <Integer,String> myPlayclone = myPlayCard.clone();
        System.out.println("Il clone è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        System.out.println("la carta e il suo clone sono uguali? :" + myPlayCard.equals(myPlayclone));
    }
}
```

Duplicazione di IMMUTABLE. Proposta2: CORRETTA

- Una definizione di clone(): È corretta?

```
class ImPairADTe<A,B> implements Cloneable{
    private final A left;//da nascondere
    private final B right;//da nascondere
    public ImPairADTe (A x, B y) {...}
    public A getLeft(){...}
    public B getRight(){...}
    public boolean equals(Object o){...}
    public ImPairADTe<A,B> clone(){
        try{return (ImPairADTe<A,B>) super.clone();}
        catch (CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

```
class main{
    public static void main(String args[]){
        ImPairADTe <Integer,String> myPlayCard = new ImPairADTe<Integer,String>(3,"fiori");
        System.out.println("la carta è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        ImPairADTe <Integer,String> myPlayclone = myPlayCard.clone();
        System.out.println("Il clone è :" + myPlayCard.getLeft() + ", " + myPlayCard.getRight());
        System.out.println("la carta e il suo clone sono uguali? :" + myPlayCard.equals(myPlayclone));
    }
}
```

- Compiliamo le due classi ed Eseguiamo (se possibile):

```
host-131-114-217-78:AltriEsercizi7Listings marcob$ javac ImPairADTe.java -d .
Note: ImPairADTe.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
host-131-114-217-78:AltriEsercizi7Listings marcob$ java IMMuPair/main
la carta è :3, fiori
Il clone è :3, fiori
la carta e il suo clone sono uguali? :true
host-131-114-217-78:AltriEsercizi7Listings marcob$
```

Duplicazione di Oggetti: MUTABLE

- **Overriding Obbligatorio** per rendere `clone`:
  - > public (usabile ovunque)
  - > calcolante duplicati "cast" sul Tipo della classe
  - > senza sollevare `CloneNotSupportedException`
  - > generante una copia shallow-intermediate-deep

- **Duplicato ottenibile sempre** da `clone()` di Object:
  - > Invocando opportunamente, `super.clone()`

- Vediamo **Caso Generale** di Definizione ed Uso:
  - > Abbiamo una classe di oggetti a cui aggiungere il metodo clone()
  - > Abbiamo una seconda classe in cui i valori della prima devono essere duplicati in uno dei 3 modi

Duplicazione di Oggetti MUTABLE: Caso Generale

- Una classe di oggetti a cui aggiungere il metodo clone(): `MuPairADTe.java`

```
import java.io.*;
import java.util.*;

class MuPairADTe <A,B> implements Cloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException{
        if(x==null||y==null) throw new invalidArgException();
        left = x; right = y;
    }
    public A getLeft(){//da aggiungere per uso effettivo
        return left;
    }
    public B getRight(){//da aggiungere per uso effettivo
        return right;
    }
    public void setLeft(A x)throws invalidArgException{//da aggiungere per uso effettivo
        if(x==null) throw new invalidArgException();
        left = x;
    }
    public void setRight(B y)throws invalidArgException{//da aggiungere per uso effettivo
        if(y==null) throw new invalidArgException();
        right = y;
    }
    //clone: inserire definizione
    public String toString(){
        return ("("+left.toString()+","+right.toString()+")");
    }
}
```

- Una seconda classe che usa e duplica i valori della prima: `main`

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
```

Duplicazione di MUTABLE. Shallow: Proposta1

- Una definizione di clone(): È corretta?

```
class MuPairADTe <A,B> implements Cloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone(){
        try{return (ImPairADTe<A,B>) super.clone();}
        catch (CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

- È corretta?: Compiliamo le due classi ed Eseguiamo (se possibile)

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
    }
}
```

Duplicazione di MUTABLE. Shallow: CORRETTA

- Una definizione di clone(): È corretta?

```
class MuPairADTe <A,B> implements Cloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone(){
        try{return (ImPairADTe<A,B>) super.clone();}
        catch (CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
    }
}
```

- Compiliamo le due classi ed Eseguiamo (se possibile):

```
host-131-114-217-78:AltriEsercizi7Listings marcob$ javac MuPairADTeS.java -d .
Note: MuPairADTeS.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
host-131-114-217-78:AltriEsercizi7Listings marcob$ java MuPair/main
l'indirizzo è via Pontecorvo, 4
Il clone è via Pontecorvo, 4
l'indirizzo è cambiato in Pontecorvo, 12
Il clone è via Pontecorvo, 4
host-131-114-217-78:AltriEsercizi7Listings marcob$
```

Duplicazione di MUTABLE. Intermediate: Proposta1

- Una definizione di clone(): È corretta?

```
class MuPairADTe <A,B> implements Cloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Intermediate su Primo componente
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof Cloneable) ret.left = ((Cloneable) left).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

- È corretta?: Compiliamo le due classi ed Eseguiamo (se possibile)

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
    }
```

Duplicazione di MUTABLE. Intermediate: Errata1

- Una definizione di clone(): È corretta?

```
class MuPairADTe <A,B> implements Cloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Intermediate su Primo componente
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof Cloneable) ret.left = ((Cloneable) left).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
    }
}
```

- Compiliamo le due classi ed Eseguiamo (se possibile):

```
host-131-114-217-78:AltriEsercizi7Listings marcob$ javac MuPairADTeIError3.java -d .
MuPairADTeIError3.java:61: error: cannot find symbol
                ret.left = ((Cloneable) left).clone();
                                              ^
  symbol:   method clone()
  location: interface Cloneable
Note: MuPairADTeIError3.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 error
host-131-114-217-78:AltriEsercizi7Listings marcob$
```

Duplicazione di MUTABLE. Intermediate: Proposta2

- Introduciamo una nuova Interfaccia public: `MuCloneable`

```
public interface MuCloneable extends Cloneable{
    public Object clone();
}

class MuPairADTe <A,B> implements MuCloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Intermediate su Primo componente
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof MuCloneable) ret.left = ((MuCloneable) left).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

- È corretta?: Compiliamo le due classi ed Eseguiamo (se possibile)

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
```

Duplicazione di MUTABLE. Intermediate: Errata2

- Una definizione di clone(): È corretta?

```
public interface MuCloneable extends Cloneable{
    public Object clone();
}

class MuPairADTe <A,B> implements MuCloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Intermediate su Primo componente
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof MuCloneable) ret.left = ((MuCloneable) left).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
    }
}
```

- Compiliamo le due classi ed Eseguiamo (se possibile):

```
host-131-114-217-78:AltriEsercizi7Listings marcob$ javac MuPairADTeIError2.java -d .
MuPairADTeIError2.java:60: error: incompatible types: Object cannot be converted to A
            if(left instanceof MuCloneable) ret.left = ((MuCloneable) left).clone();
                                                                           ^
  where A is a type-variable:
    A extends Object declared in class MuPairADTe
Note: MuPairADTeIError2.java uses unchecked or unsafe operations.
```

Duplicazione di MUTABLE. Intermediate: Modifica

- Introduciamo una nuova Interfaccia public: `MuCloneable`

```
public interface MuCloneable extends Cloneable{
    public Object clone();
}

class MuPairADTe <A,B> implements MuCloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Intermediate su Primo componente
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof MuCloneable) ret.left = (A)((MuCloneable) left).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

- È corretta?: Compiliamo le due classi ed Eseguiamo (se possibile)

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
```

Duplicazione di MUTABLE. Intermediate: Corretta

- Una definizione di clone(): È corretta?

```
public interface MuCloneable extends Cloneable{
    public Object clone();
}

class MuPairADTe <A,B> implements MuCloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Intermediate su Primo componente
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof MuCloneable) ret.left = (A)((MuCloneable) left).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
    }
}
```

- Compiliamo le due classi ed Eseguiamo (se possibile):

```
host-131-114-217-78:AltriEsercizi7Listings marcob$ javac MuPairADTeI.java -d .
Note: MuPairADTeI.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
host-131-114-217-78:AltriEsercizi7Listings marcob$ java MuPair/main
l'indirizzo è via Pontecorvo, 4
Il clone è via Pontecorvo, 4
l'indirizzo è cambiato in Pontecorvo, 12
Il clone è via Pontecorvo, 4
host-131-114-217-78:AltriEsercizi7Listings marcob$
```

Duplicazione di MUTABLE. Deep: Soluzione

- Una definizione di clone(): È corretta

```
class MuPairADTe <A,B> implements MuCloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Deep
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof MuCloneable) ret.left = (A)((MuCloneable) left).clone();
            if(right instanceof MuCloneable) ret.right = (B)((MuCloneable) right).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

- È corretta?: Compiliamo le due classi ed Eseguiamo (se possibile)

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
```

Duplicazione di MUTABLE. Deep: Corretta

- Una definizione di clone(): È corretta?

```
class MuPairADTe <A,B> implements MuCloneable{
    private A left;//da nascondere
    private B right;//da nascondere
    public MuPairADTe(A x, B y) throws invalidArgException {...}
    public A getLeft(){...}
    public B getRight(){...}
    public void setLeft(A x)throws invalidArgException{...}
    public void setRight(B y)throws invalidArgException{...}
    public MuPairADTe<A,B> clone() {//Deep
        try{MuPairADTe<A,B> ret = (MuPairADTe<A,B>) super.clone();
            if(left instanceof MuCloneable) ret.left = (A)((MuCloneable) left).clone();
            if(right instanceof MuCloneable) ret.right = (B)((MuCloneable) right).clone();
            return ret;}
        catch(CloneNotSupportedException e){return null;}
    }
    public String toString(){...}
}
```

```
class main{
public static void main(String args[]) throws invalidArgException{
MuPairADTe <String,Integer> myAddress =
                    new MuPairADTe<String,Integer>("Pontecorvo",4);
System.out.println("l'indirizzo è via " + myAddress.getLeft() + ", " + myAddress.getRight());
MuPairADTe <String,Integer> myAddClone = myAddress.clone();
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
myAddress.setRight(12);
System.out.println("l'indirizzo è cambiato in " + myAddress.getLeft() + ", " + myAddress.getRight());
System.out.println("Il clone è via " + myAddClone.getLeft() + ", " + myAddClone.getRight());
    }
}
```

- Compiliamo le due classi ed Eseguiamo (se possibile):

```
host-131-114-217-78:AltriEsercizi7Listings marcob$ javac MuPairADTeD.java -d .
Note: MuPairADTeD.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
host-131-114-217-78:AltriEsercizi7Listings marcob$ java MuPair/main
l'indirizzo è via Pontecorvo, 4
Il clone è via Pontecorvo, 4
l'indirizzo è cambiato in Pontecorvo, 12
Il clone è via Pontecorvo, 4
host-131-114-217-78:AltriEsercizi7Listings marcob$
```

Presentazione dei valori: toString

- metodo **toString**:
  - Definito in Object per tutte le classi
    ```
    public String toString()
    ```
  - Crea una stringa che presenta l'oggetto in modo testuale
  - Overidden: per fornire una presentazione dei valori

ImPairADTe con equals, clone, toString e Caso di uso

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

class main{
  public static void main(String args[]){
    ImPairADTe <Integer,String> myPlayCard = new ImPairADTe<Integer,String>(3,"fiori");
    System.out.println("la carta è :" + myPlayCard);
    ImPairADTe <Integer,String> myPlayclone = myPlayCard.clone();
    System.out.println("Il clone è :" + myPlayclone);
    System.out.println("Carta e clone sono uguali? :" + myPlayCard.equals(myPlayclone));
    }
}
```

Collezione degli elementi: elements

- 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 Vector<T> elements()
    ```

Collezione degli elementi: elements /2

- metodo **elements**:
  - Permette di aumentare l'**usabilità** dei valori astratti proteggendone l'**integrità**

    vedi caso di uso in class Main di file `muSetADTX.java` più avanti (in collection e enhanced for)

Definition (Integrità di Valore o Dato)

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

Valori Collection e enhanced for

- Interfaccia Collection è per valori che esprimono:
  - Collezioni di valori
  - Sono superTipi 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.

- vedi caso di uso in file `muSetADTX.java` accluso.

elements e enhanced for: Usati insieme

```
import java.lang.*;
import java.util.*;

public interface Elements<T>{
    public  Vector<T> elements();
}

class MuSetADTX<T> implements MuCloneable,Elements<T>{
    private boolean empty;
    private T elem;
    private MuSetADTX<T> rest;
//metodi
    public MuSetADTX(){...}
    public void add (T x) {...}
    public void remove (T x) {...}
    public boolean isEmpty () {...}
    public boolean isIn (T x) {...}
    public int size () {...}
//additionals
    ...
    public Vector<T> elements() {...}
}
class Main{
    public static void main(String args[]){
        MuSetADTX<Integer> aSet = new MuSetADTX<T>();
        aSet.add(3);
        ...
        if (!aSet.isEmpty()){
            int maxaSet = 0;
            for(Integer n: aSet.elements()){
                if(n>maxaSet)maxaSet=n;
            }
            ...
    }
```