1) Tutti gli oggetti di classe T predefinita che estendono Cloneable hanno un metodo

public Object clone()

che overridden il metodo clone di Object:

protected Object clone() throws CloneNotSupportedException

ed ha il seguente comportamento.

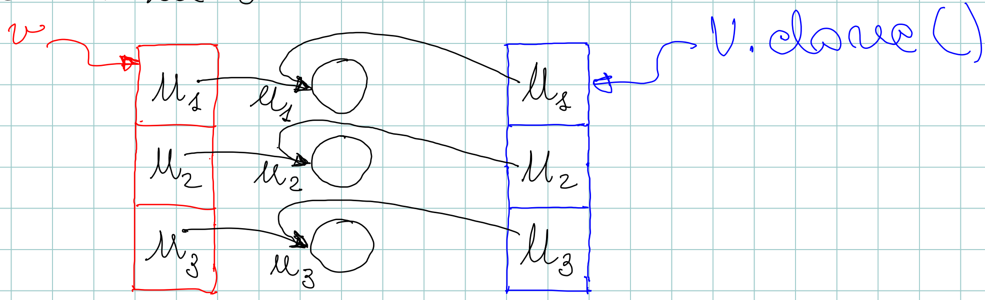
Sia $v \in [T]$. Allora:

1. v.clone() $\in$ [T]
2. v != v.clone()
3. v.equals(v.clone())
4. v.clone() è ottenuta
   - creando un nuovo oggetto che indichiamo con V_{clone}, ovvero: T V_{clone} = v.clone();
   - copiando il contenuto di ogni field di v nel corrispondente field del nuovo oggetto:

     $\forall$ field:

     V_{clone}.field == v.field

Nel caso di un oggetto Vector<U> v, dove U sia un tipo e $u_1, u_2, u_3 \in [U]$, i 3 valori contenuti in v



2) Vediamo la metodologia più semplice per definire il metodo clone di nuove classi che implementano Cloneable.

La definizione cambia secondo che siano classi per valori IMMUTABLE oppure MUTABLE

2.1. Classi IMMUTABLE.

Procediamo esattamente come per le classi predisposte di tipo Cloneable.
Sia A la nostra classe. Siano $F_1, \dots, F_k$ i field di istanza di A. Allora

```
MA class A implements Cloneable {
    // qui Fields e Metodi di A
    public A clone() {
        A ret = new A();
        ret.F1 = F1;
        :
        ret.Fk = Fk;
        return ret;
    } // altri obbligatorios
}
```

dove MA sia il nostro blocco classe A.

2.2 Classi MUTABLE.

Diversamente dalle classi cloneable per IMMUT., qui, in aggiunta allo stato, duplichiamo il valore di ogni Field, con lo scopo di soddisfare V.equals(V.clone())

Sia A la nuova classe. Siano $F_1, \dots, F_k$ i field di interfaccia di A. Allora

```
M_A class A implements Cloneable{
    // qui Fields e Metodi di A

    public A clone(){
        A ret = new A();
        ret.F1 = F1.clone();
        :
        ret.Fk = Fk.clone();
        return ret;
    }
    // altri altri metodi
}
```

dove M_A sia il modificatore della classe A.

Qui abbiamo assunto che tutti i Fields siano di tipo Cloneable. Per quei Field che abbiamo tipo non cloneable bisogna procedere "ad hoc", distinguendo tra valore di:

+ tipo non cloneable ma a stato accessibile e di cui posso costruire una copia ad hoc:

  $B_{F_i}\ temp_{F_i} = new\ B_{F_i}();$

  // duplicazione dello stato del valore del Field F_i di tipo B_{F_i} con stato accessibile

+ tipo non cloneable e con stato private o non accessibile da A.

  In questo caso non possiamo duplicare

  $ret.F_i = F_i$

  Dobbiamo contare sulla presenza di almeno 1 field duplicabile.