```
/*
 Uso di struttura condivisa per supportare valori a componenti non modificabili, e a
 struttura dinamica. Tipico modo di realizzare i valori nei Linguaggi Funzionali.
 CONSIGLIATO: Semplice ed efficiente, evita di copiare strutture.
*/


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

interface RelazioneAPI <T1,T2>{ //One Java API for the type Relazione Immutable
    public boolean isIn1 (T1 e);
    public Vector<T2> getAll2(T1 e);
    public RelazioneAPI<T1,T2> add(T1 x, T2 y);
}

class RelazioneADT<T1,T2> implements RelazioneAPI<T1,T2>{
    private T1 left;
    private T2 right;
    private RelazioneAPI<T1,T2> rem;

    public RelazioneADT(){
    }
    public RelazioneADT(T1 x, T2 y, RelazioneAPI<T1,T2> r){
        left = x; right = y; rem = r;
    }
    public boolean isIn1 (T1 e){
        return (left != null && (left.equals(e)||rem.isIn1(e)));
    }
    public Vector<T2> getAll2(T1 e){
        if (left == null) return new Vector<T2>();
        Vector<T2> r = rem.getAll2(e);
        if (left.equals(e)) r.add(right);
        return r;
    }
    public RelazioneAPI<T1,T2> add(T1 x, T2 y){
        return new RelazioneADT<T1,T2>(x,y,this);
    }
}

/*
 Esercizio 3.
 Siano RelazioneAPI un API per relazioni con le operazioni definite come sopra,
 e RelazioneADT un ADT che la implementa. Si estenda RelazioneADT per definire
 un tipo FunRel per relazioni che sono funzioni finite su generici tipi ed aventi,
 in aggiunta alle operazioni di RelazioneAPI, le operazioni:
    - addFun che opera come add ma solleva eccezione AlreadyDefined se si tenta di
            cambiare la immagine di un valore definito.
    - apply che applicata ad un valore restituisce l'immagine della funzione se
            definita su tale valore, altrimenti solleva UndefinedException.
 */

class UndefinedException extends Exception{
    UndefinedException(String x){
        super(x);
    }
}
class AlreadyDefinedException extends Exception{
    AlreadyDefinedException(String x){
        super(x);
```

```
    }
}

class FunRel<T1,T2> extends RelazioneADT<T1,T2>{

    public RelazioneAPI<T1,T2> add(T1 x, T2 y){
        Vector<T2> im = getAll2(x);
        if (im.size()==0)return super.add(x,y);
        return this;
    }
    public T2 apply(T1 x) throws UndefinedException{
        Vector<T2> im = getAll2(x);
        if (im.size()==0) throw new UndefinedException("apply");
        return im.get(0);
    }
}
/* La classe FunRel<T1,T2> definita sopra, è compilata senza rilevare errori, ma non
   è una soluzione al problema di esercizio 3.
   La ragione è nella definizione data per il metodo add. In accordo a tale definizione
   add genera un valore RelazioneADT<T1,T2> che non è valore FunRel<T1,T2>.
   Come mai?
   Rispondere, Correggere, Compilare ed Applicare la corretta soluzione al successivo
   esercizio5
*/

/* Esecizio 5.
 Mostrare un programma che utilizza le classi sopra definite per:
 1- definire una funzione ff: ff(2)=2, ff(3)=6. Quindi,
 2- applicare ad essa le operazioni opportune per:
    - calcolare e stampare ff(3)
    - calcolre getAll2 di ff con 3
    - estendere ff con ff(2)=5
    - calcolare e stampare ff(2)

*/

class Main{
    public static void main(String[] in){
        FunRel<Integer,Integer> ff = new FunRel<Integer,Integer>();
        //...
    }
}
```