```
/*
 Esercizio1.
 Sia STree un tipo astratto per alberi MODIFICABILI, con nodi interni etichettati con tipo
 generico A e nodi foglia etichettati con tipo generico B. In aggiuta ai costruttori per
 albero vuoto e non, STree fornisce le seguenti operazioni:
    isEmpty [isRoot, is Leaf] che, applicata ad un STree s, restituisce true se s è l'abero
        vuoto [ha radice, è foglia], false altrimenti;
    getRoot che, applicata ad un STree s, restituisce l'etichetta della radice, se s ha ra-
        dice, eccezione altrimenti;
    getSons che, applicata ad un STree s, restituisce il vettore degli alberi figli della
        radice, se s ha radice, eccezione altrimenti;
    getLeaf che, applicata ad un STree s, restituisce l'etichetta del nodo se s è una fogli
        eccezione altrimenti;
    update che, applicata ad un STree s, ad un cammino p dalla radice, e ad un albero r,
        rimpiazza il sottoalbero al cammino p, se esiste ed è un albero con radice, con l'a
        bero r. Solleva opportune eccezioni se tale sottoalbero non esiste, ovvero non è un
        albero con radice.
 Si fornisca un API ed un ADT (con le sole implementazioni di ...)
 Allo scopo si utilizzino Vector<Integer> per cammini, dove n::ns dato un albero
 <u - t1 ... tn ... tm> indica il cammino ns del sottoalbero tn.
 */


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


class NoRootException extends Exception{
    public NoRootException(String s){super(s);}
}
class NoLeafException extends Exception{
    public NoLeafException(String s){super(s);}
}
class InvalidArgumentException extends Exception{
    public InvalidArgumentException(String s){super(s);}
}

//Si mostri l'API
interface STree <A,B> {
    public boolean isEmpty();
    public boolean isRoot();
    public boolean isLeaf();
    public A getRoot() throws NoRootException;
    public Vector<? extends STree<A,B>> getSons() throws NoRootException;
    public B getLeaf() throws NoLeafException;
    public void update(Vector<Integer> p, STree <A,B> r)
                            throws NoRootException,InvalidArgumentException;
}

/*
 Si mostrino le definizioni di:
    - stato
    - costruttore di un albero con radice,
    - operazione update
 Si utilizzi implementazione singola
*/

class Astree <A,B> implements STree <A,B> {
    private A root;
    private B leaf;
```

```
    private Vector<? extends STree<A,B>> sons;
    Astree(){
    }
    Astree(B leaf)throws InvalidArgumentException{
        if(leaf!=null){
            this.leaf = leaf;
            return;
        }
        throw new InvalidArgumentException("constructor: leaf is null");
    }
    Astree(A root, Vector<? extends STree<A,B>> sons)throws InvalidArgumentException{
        if((root!=null)&&(sons!=null)){
            this.root = root; this.sons = sons;
            return;
        }
        throw new InvalidArgumentException("constructor: root/sons is null");
    }
    public boolean isEmpty(){
        return (root==null&&leaf==null&&sons==null);
    }
    public boolean isRoot(){
        return (root!=null);
    }
    public boolean isLeaf(){
        return (leaf!=null);
    }
    public A getRoot() throws NoRootException{
        if (root!=null) return root;
        throw new NoRootException("getRoot");
    }
    public B getLeaf() throws NoLeafException{
        if (leaf!=null) return leaf;
        throw new NoLeafException("getLeaf");
    }
    public Vector<? extends STree<A,B>> getSons() throws NoRootException{
        if (root!=null) return sons;
        throw new NoRootException("getSons");
    }
    public void update(Vector<Integer> p, STree <A,B> r)throws NoRootException,
                                                               InvalidArgumentException{
        if (isRoot()&&(p.size()==0)) {
            root = ((Astree<A,B>)r).root;
            sons = ((Astree<A,B>)r).sons;
            return;
        }
        if (isRoot()) {
            int pHead = p.get(0).intValue();
            if (sons.size()<pHead) throw new InvalidArgumentException("updade: path");
            STree <A,B> theSon = sons.get(pHead-1);
            p.remove(0);
            theSon.update(p,r);
            return;
        }
        throw new NoRootException("updade");
    }
}
```