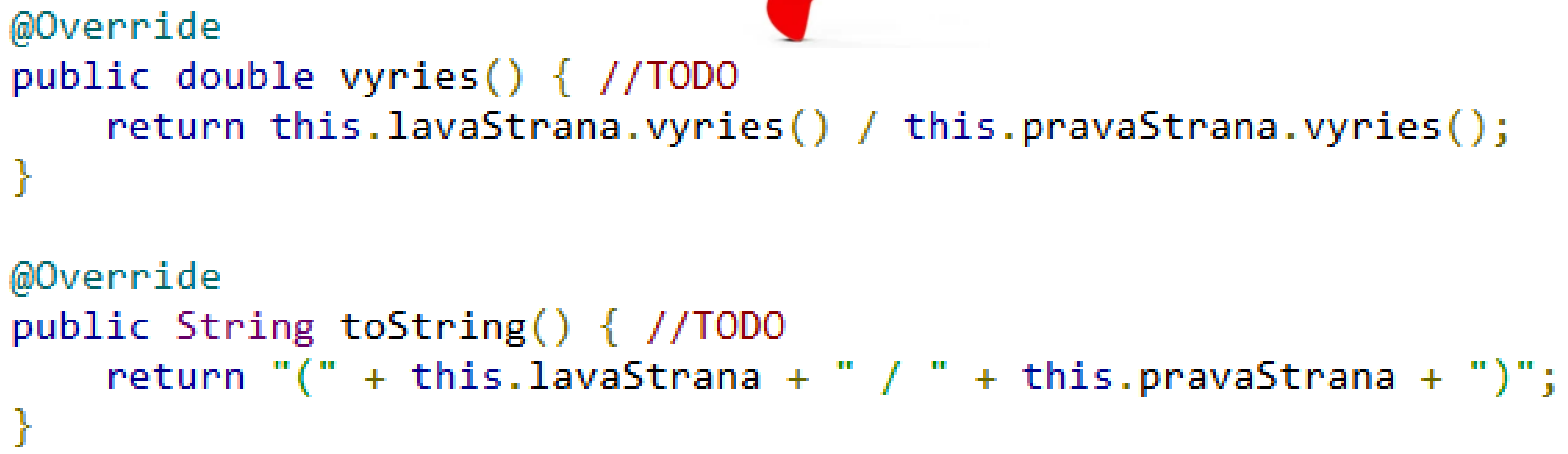
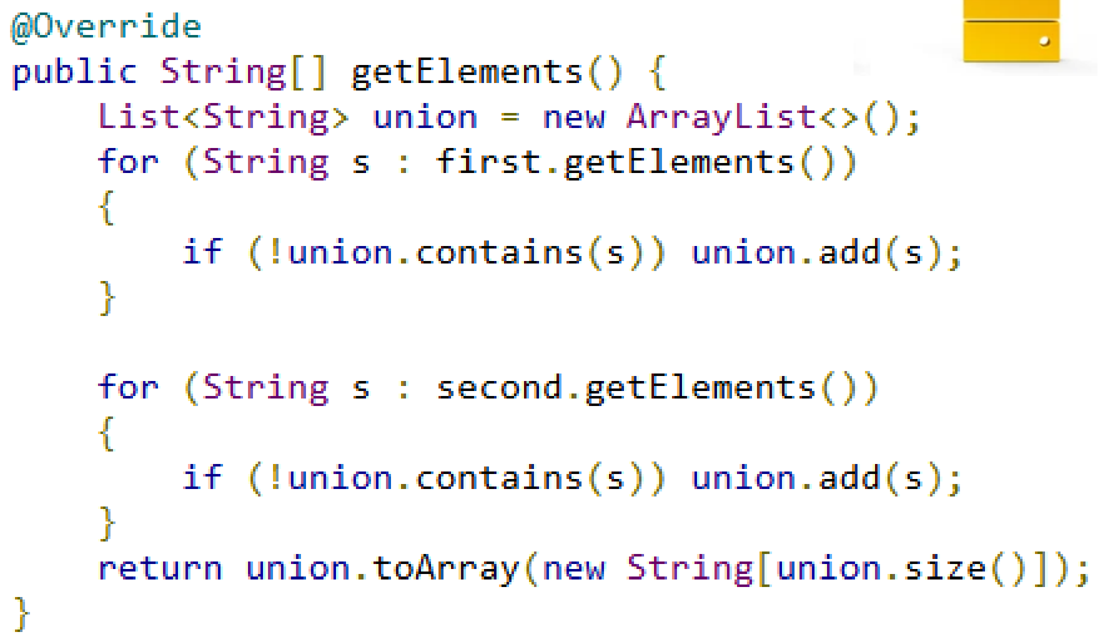
TODO or not TODO
If you do something remove TODO.



```
@Override
public double vyries() { //TODO
    return this.lavaStrana.vyries() / this.pravaStrana.vyries();
}

@Override
public String toString() { //TODO
    return "(" + this.lavaStrana + " / " + this.pravaStrana + ")";
}
```

Je lepšie volať contains() na množine ako na liste.(aj keď naša implementácia nie je efektívna)



```java
@Override
public String[] getElements() {
    List<String> union = new ArrayList<>();
    for (String s : first.getElements())
    {
        if (!union.contains(s)) union.add(s);
    }

    for (String s : second.getElements())
    {
        if (!union.contains(s)) union.add(s);
    }
    return union.toArray(new String[union.size()]);
}
```

Use what you have.

```
public boolean add(String x) {
        if(last >= Capacity) {
                return false;
        }
        for (int i = 0; i < last; i++) {
                if (elements[i] == null && x == null) {
                        return false;
                }
                if (elements[i] != null && elements[i].equals(x)) {
                        return false;
                }
        }
        elements[last++] = x;
        return true;
}
```

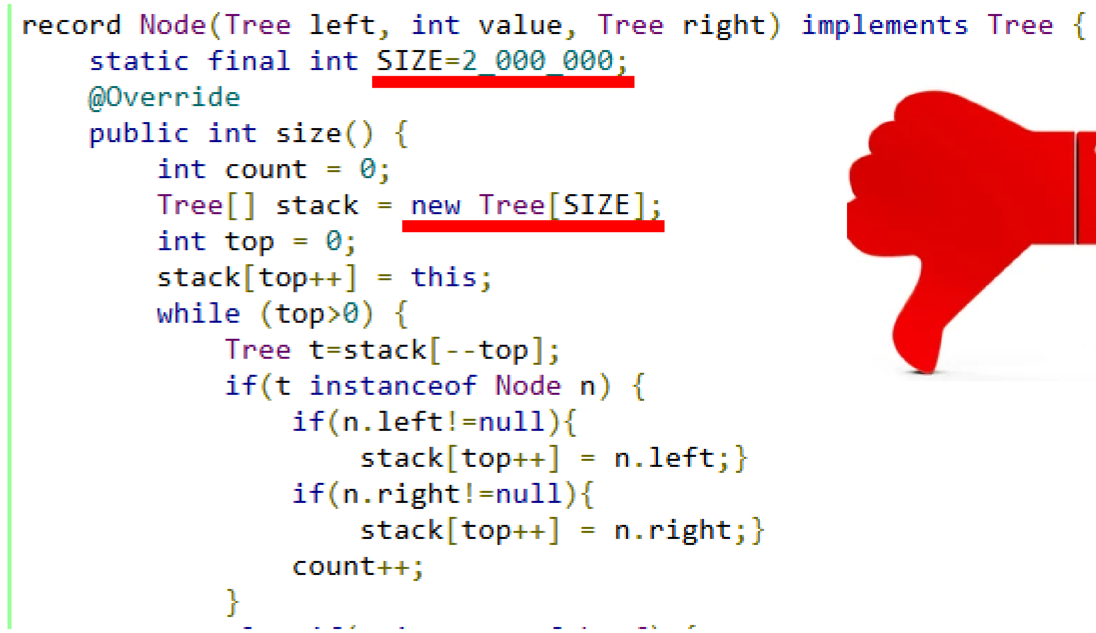
Contains()

Use what you have.

```java
@Override
public String[] getElements() {
        if (left == null || right == null) {
                String[] out = {};
                return out;
        }

        String[] leftElements = left.getElements();

        StringSet out = new StringSet(left.size());

        for (String element : leftElements) {
                if (right.contains(element)) {
                        out.add(element);
                }
        }

        return out.getElements();
```

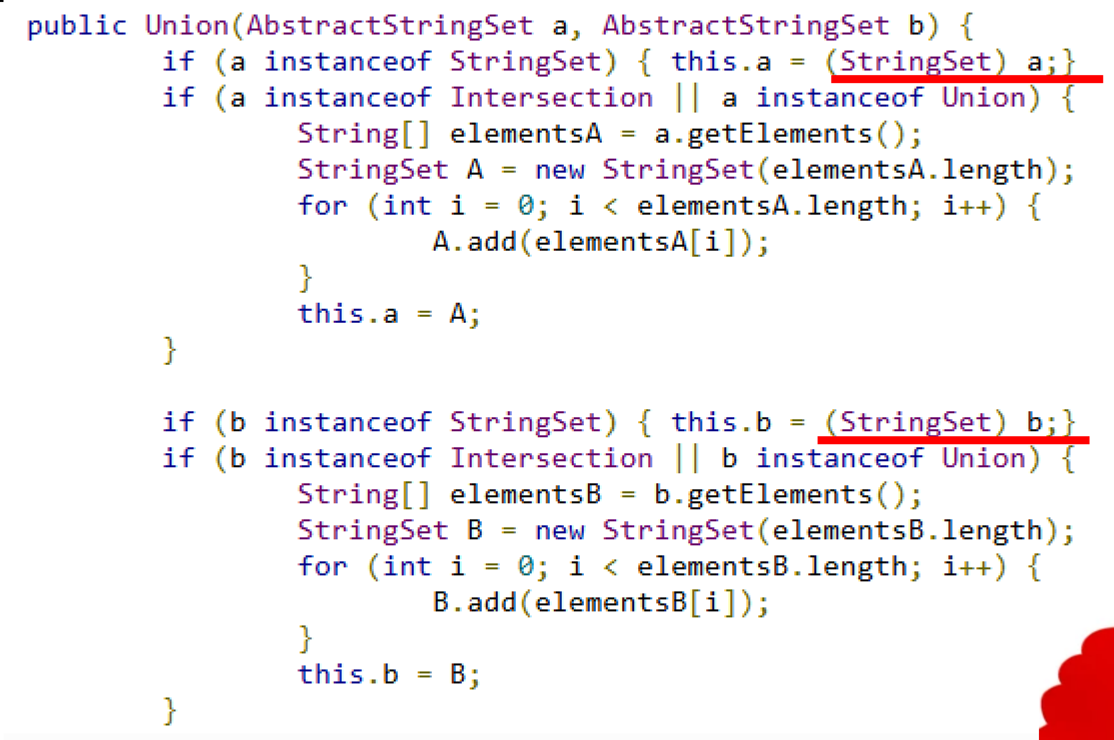
```java
public String[] getElements() {
        StringSet temp = new StringSet(a.size() + b.size());
        int i = 0;
        for (String s : a.getElements()) {
                temp.add(s);
        }
        for (String s : b.getElements()) {
                temp.add(s);
        }
        return temp.getElements();
}
```

Stack s fixnou veľkosťou nie je šťastné riešenie.



```
record Node(Tree left, int value, Tree right) implements Tree {
    static final int SIZE=2_000_000;
    @Override
    public int size() {
        int count = 0;
        Tree[] stack = new Tree[SIZE];
        int top = 0;
        stack[top++] = this;
        while (top>0) {
            Tree t=stack[--top];
            if(t instanceof Node n) {
                if(n.left!=null){
                    stack[top++] = n.left;}
                if(n.right!=null){
                    stack[top++] = n.right;}
                count++;
            }
```

This.a používa abstraktnú triedu ->netreba pretypovať. Rovnako aj po pattern matching.



```java
public Union(AbstractStringSet a, AbstractStringSet b) {
        if (a instanceof StringSet) { this.a = (StringSet) a;}
        if (a instanceof Intersection || a instanceof Union) {
                String[] elementsA = a.getElements();
                StringSet A = new StringSet(elementsA.length);
                for (int i = 0; i < elementsA.length; i++) {
                        A.add(elementsA[i]);
                }
                this.a = A;
        }

        if (b instanceof StringSet) { this.b = (StringSet) b;}
        if (b instanceof Intersection || b instanceof Union) {
                String[] elementsB = b.getElements();
                StringSet B = new StringSet(elementsB.length);
                for (int i = 0; i < elementsB.length; i++) {
                        B.add(elementsB[i]);
                }
                this.b = B;
        }
```