i219 Software Design Methodology
7. Information hiding and reuse

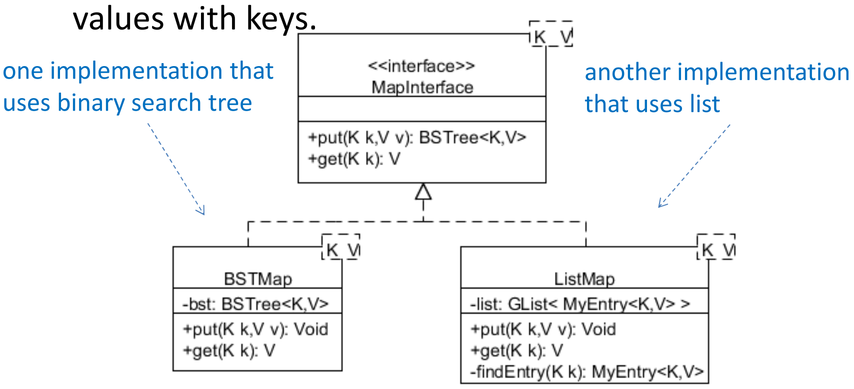
Kazuhiro Ogata (JAIST)

Outline of lecture

- Information hiding
- Reuse
- Some examples for information hiding & reuse
  - Map
  - Parse tree for expression
  - Parse tree for assignment programs

Information hiding

- Let us design a functionality (called map) that associates values with keys.



one implementation that uses binary search tree

another implementation that uses list

K, V

<<interface>>
MapInterface

+put(K k,V v): BSTree<K,V>
+get(K k): V

K, V

BSTMap

-bst: BSTree<K,V>

+put(K k,V v): Void
+get(K k): V

K, V

ListMap

-list: GList< MyEntry<K,V> >

+put(K k,V v): Void
+get(K k): V
-findEntry(K k): MyEntry<K,V>

Both classes hide how to implement the functionality but just provide the functionality (the interface) – information hiding.

Reuse

In the following program, BSTMap<K,V> is used. But, the only part that depends on the class is BSTMap:

```
MapInterface<String,Integer> map = new BSTMap<String,Integer>();
map.put("y",3);
map.put("x",4);
map.put("z",5);
map.put("x",6);
System.out.println(map.get("x"));
```

can be replaced with ListMap

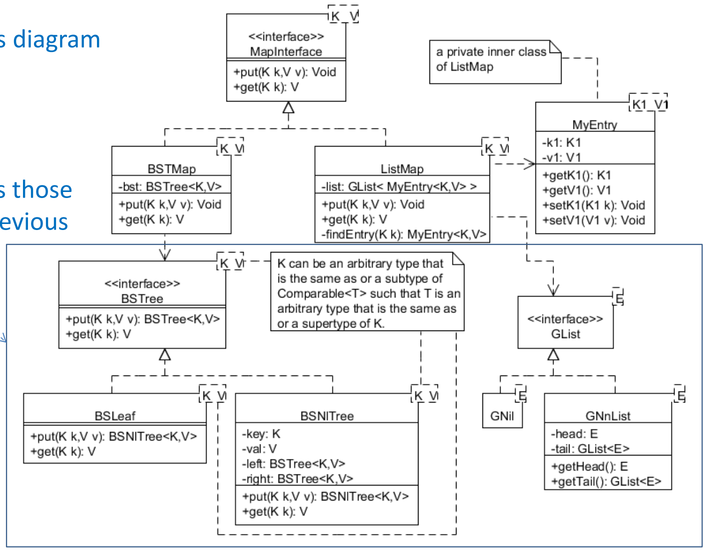
Use of implementations via an interface makes it possible to make dependent part local, increasing reusability.

If you want to use ListMap<K,V> instead of BSTMap<K,V>, all you have to do is to replace BSTMap with ListMap.

Map (1)

a detailed class diagram on map

almost the same as those discussed in the previous lecture



K V
<<interface>>
MapInterface
+put(K k,V v): Void
+get(K k): V

a private inner class of ListMap

K1 V1
MyEntry
-k1: K1
-v1: V1
+getK1(): K1
+getV1(): V1
+setK1(K1 k): Void
+setV1(V1 v): Void

K V
BSTMap
-bst: BSTree<K,V>
+put(K k,V v): Void
+get(K k): V

K V
ListMap
-list: GList< MyEntry<K,V> >
+put(K k,V v): Void
+get(K k): V
-findEntry(K k): MyEntry<K,V>

K V
<<interface>>
BSTree
+put(K k,V v): BSTree<K,V>
+get(K k): V

K can be an arbitrary type that is the same as or a subtype of Comparable<T> such that T is an arbitrary type that is the same as or a supertype of K.

E
<<interface>>
GList

K V
BSLeaf
+put(K k,V v): BSNITree<K,V>
+get(K k): V

K V
BSNITree
-key: K
-val: V
-left: BSTree<K,V>
-right: BSTree<K,V>
+put(K k,V v): BSNITree<K,V>
+get(K k): V

E
GNil

E
GNnList
-head: E
-tail: GList<E>
+getHead(): E
+getTail(): GList<E>

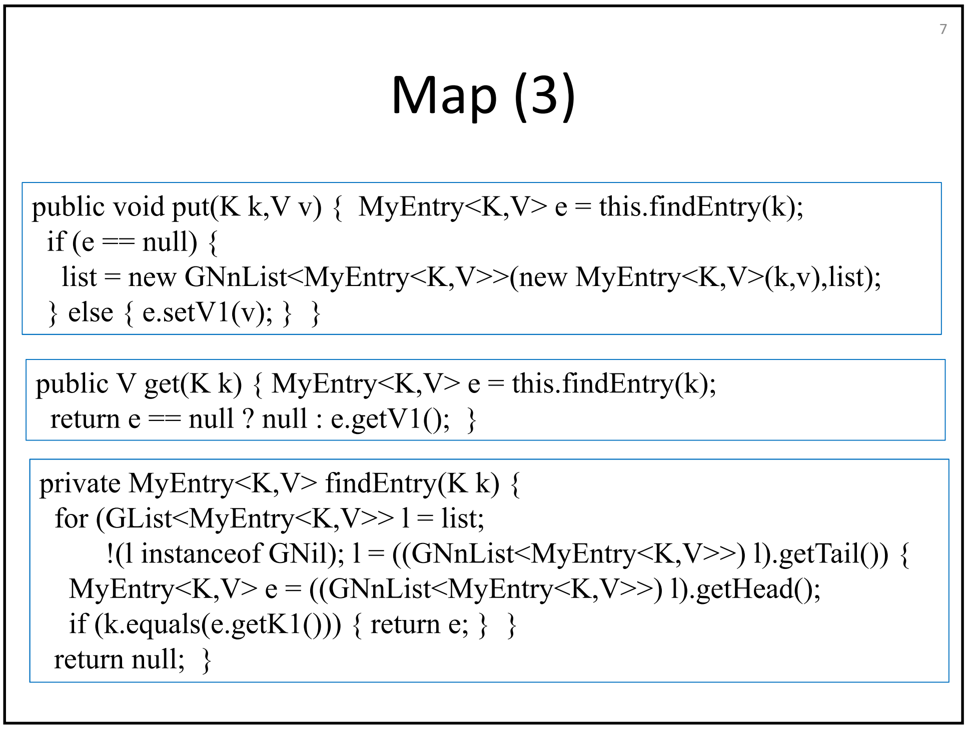
Map (2)

```
public class ListMap<K,V> implements MapInterface<K,V> {
  private class MyEntry<K1,V1> { … }
  private GList<MyEntry<K,V>> list;
  public ListMap() { list = new GNil<MyEntry<K,V>>(); }
  public void put(K k,V v) { … }
  public V get(K k) { … }
 private MyEntry<K,V> findEntry(K k) { … }
}
```

an inner class

list is initialized as nil (the empty list)

```
private K1 k1;  private V1 v1;
public MyEntry(K1 k1,V1 v1) { this.k1 = k1; this.v1 = v1; }
public K1 getK1() { return k1; }
public V1 getV1() { return v1; }
public void setK1(K1 k1) { this.k1 = k1; }
public void setV1(V1 v1) { this.v1 = v1; }
```

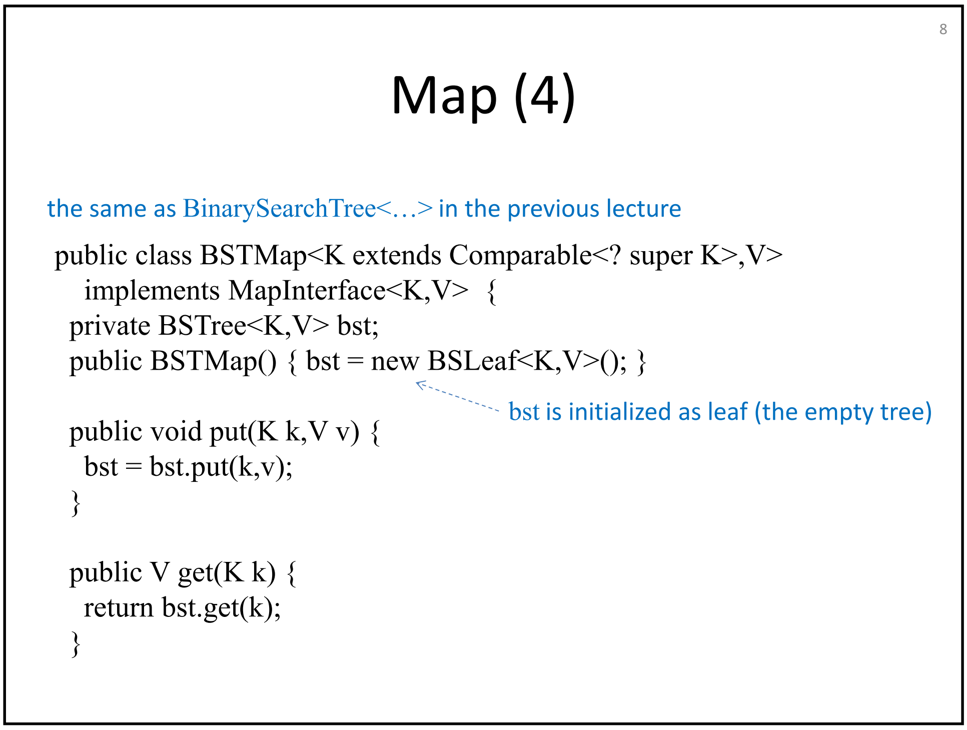


Map (3)

```
public void put(K k,V v) {  MyEntry<K,V> e = this.findEntry(k);
  if (e == null) {
    list = new GNnList<MyEntry<K,V>>(new MyEntry<K,V>(k,v),list);
  } else { e.setV1(v); }  }
```

```
public V get(K k) { MyEntry<K,V> e = this.findEntry(k);
  return e == null ? null : e.getV1();  }
```

```
private MyEntry<K,V> findEntry(K k) {
  for (GList<MyEntry<K,V>> l = list;
       !(l instanceof GNil); l = ((GNnList<MyEntry<K,V>>) l).getTail()) {
    MyEntry<K,V> e = ((GNnList<MyEntry<K,V>>) l).getHead();
    if (k.equals(e.getK1())) { return e; }  }
  return null;  }
```



Map (4)

the same as BinarySearchTree<…> in the previous lecture

```
public class BSTMap<K extends Comparable<? super K>,V>
    implements MapInterface<K,V>  {
  private BSTree<K,V> bst;
  public BSTMap() { bst = new BSLeaf<K,V>(); }
```

bst is initialized as leaf (the empty tree)

```
  public void put(K k,V v) {
    bst = bst.put(k,v);
  }

  public V get(K k) {
    return bst.get(k);
  }
```

Map (5)

Map is provided in Java libraries.

an interface provided by Java libraries

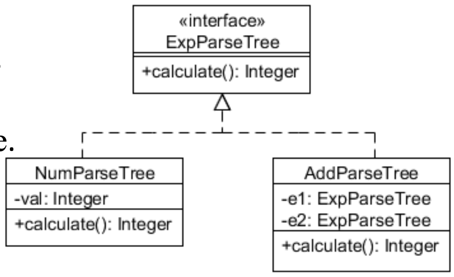
an implementation of Map provided by Java libraries

```
Map<String,Integer> map = new HashMap<String,Integer>();
map.put("y",3);
map.put("x",4);
map.put("z",5);
map.put("x",6);
System.out.println(map.get("x"));
```

Package java.util.* needs to be imported as “import java.util.*;”.

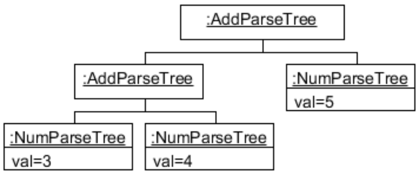
Parse tree for expression (1)

Any expression that consists of integers, addition and parentheses can be represented as an instance of NumParseTree or AddParseTree.

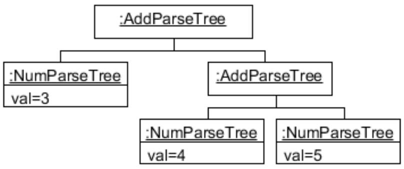


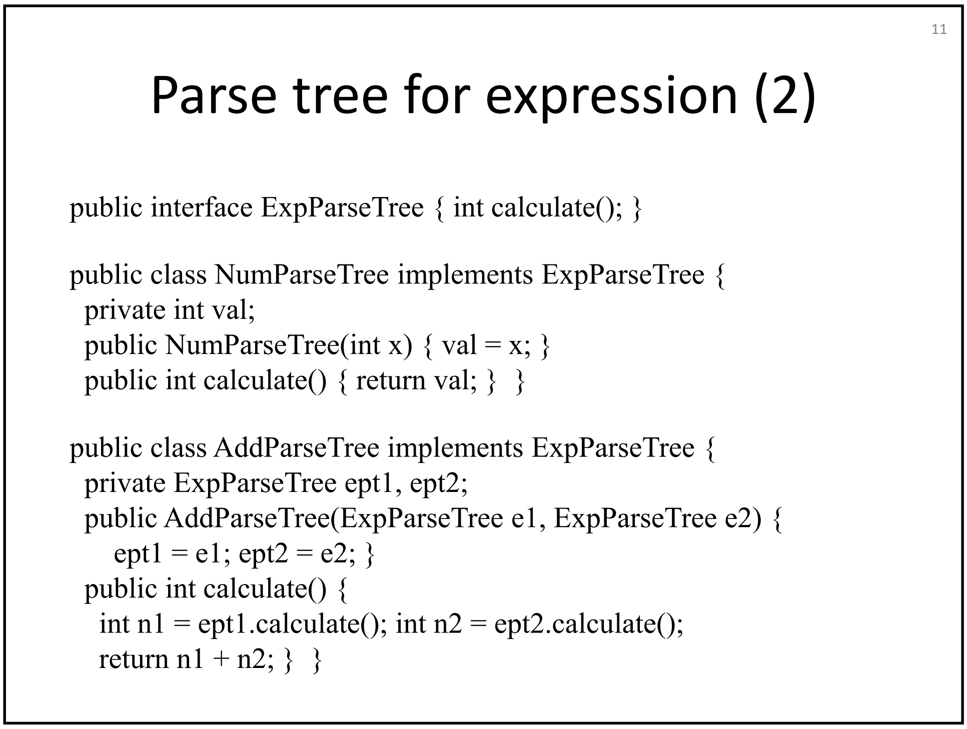
✓ val, e1 & e2 are private, and hidden.
✓ NumParseTree & AddParseTree can be used through ExpParseTree.

(3+4)+5 (or 3+4+5)



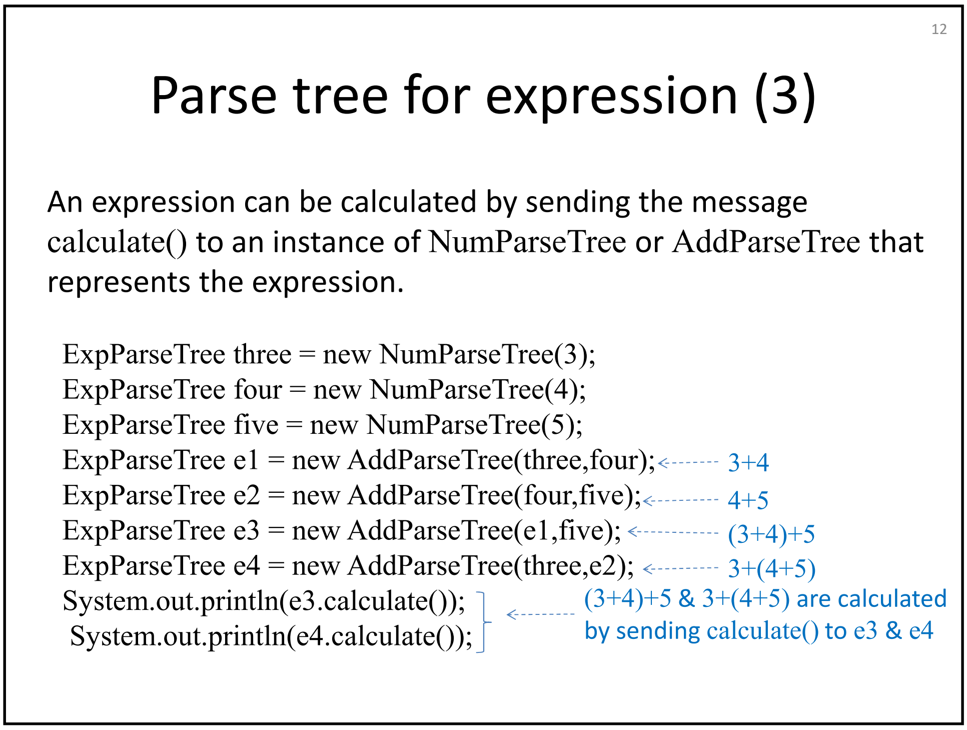
3+(4+5)





Parse tree for expression (2)

```
public interface ExpParseTree { int calculate(); }

public class NumParseTree implements ExpParseTree {
  private int val;
  public NumParseTree(int x) { val = x; }
  public int calculate() { return val; }  }

public class AddParseTree implements ExpParseTree {
  private ExpParseTree ept1, ept2;
  public AddParseTree(ExpParseTree e1, ExpParseTree e2) {
      ept1 = e1; ept2 = e2; }
  public int calculate() {
    int n1 = ept1.calculate(); int n2 = ept2.calculate();
    return n1 + n2; }  }
```

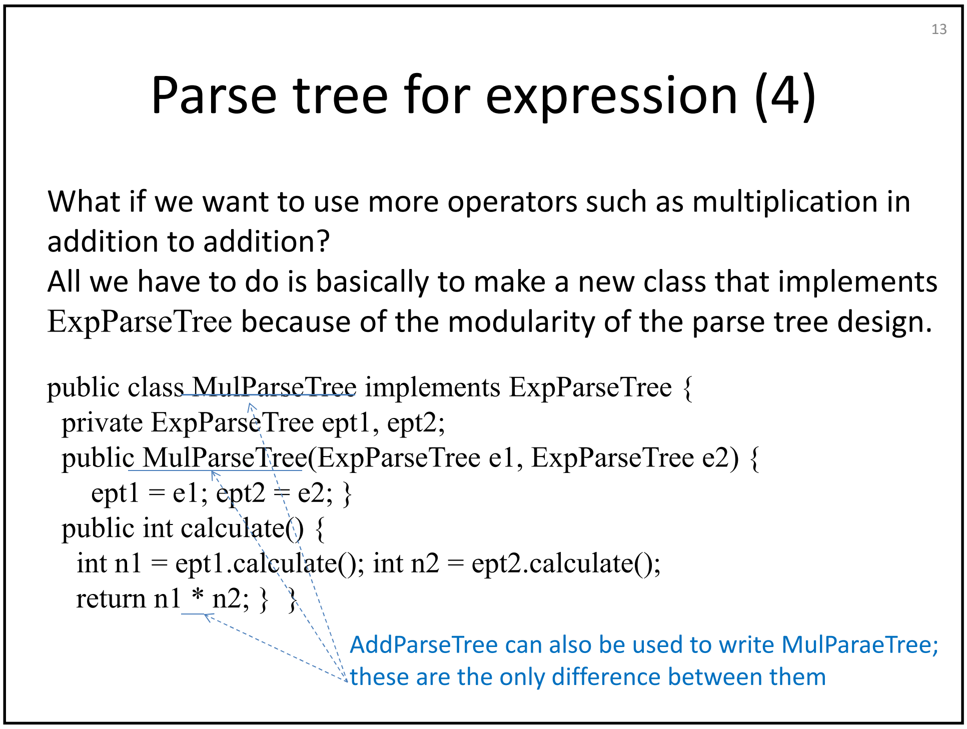


Parse tree for expression (3)

An expression can be calculated by sending the message calculate() to an instance of NumParseTree or AddParseTree that represents the expression.

```
ExpParseTree three = new NumParseTree(3);
ExpParseTree four = new NumParseTree(4);
ExpParseTree five = new NumParseTree(5);
ExpParseTree e1 = new AddParseTree(three,four);        3+4
ExpParseTree e2 = new AddParseTree(four,five);         4+5
ExpParseTree e3 = new AddParseTree(e1,five);           (3+4)+5
ExpParseTree e4 = new AddParseTree(three,e2);          3+(4+5)
System.out.println(e3.calculate());
 System.out.println(e4.calculate());
```

(3+4)+5 & 3+(4+5) are calculated by sending calculate() to e3 & e4

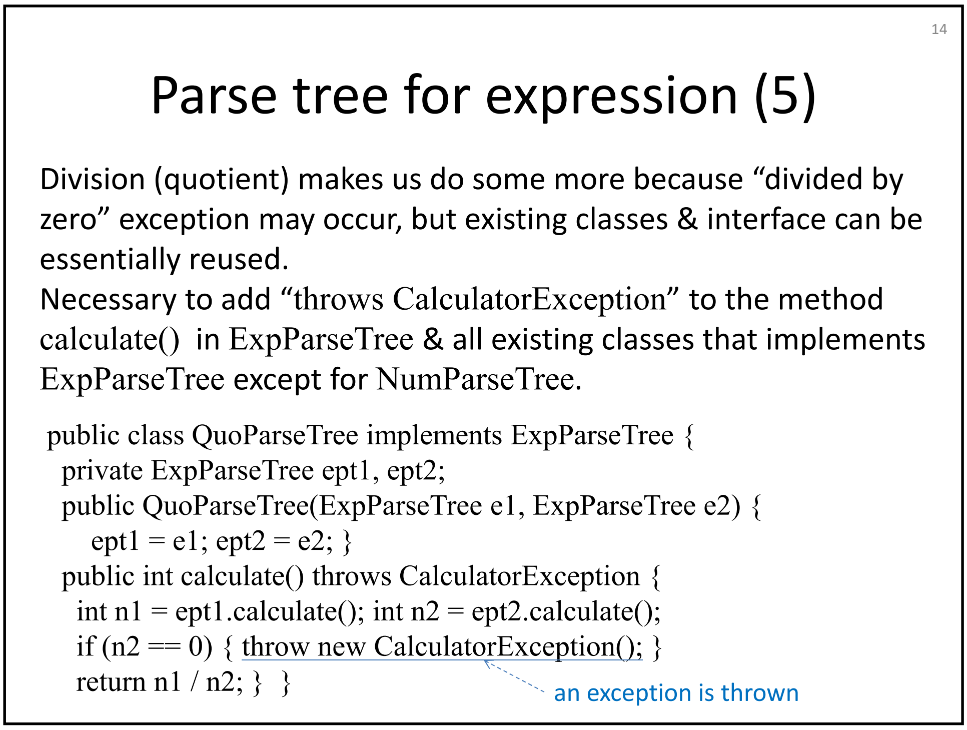


Parse tree for expression (4)

What if we want to use more operators such as multiplication in addition to addition?
All we have to do is basically to make a new class that implements ExpParseTree because of the modularity of the parse tree design.

```
public class MulParseTree implements ExpParseTree {
  private ExpParseTree ept1, ept2;
  public MulParseTree(ExpParseTree e1, ExpParseTree e2) {
      ept1 = e1; ept2 = e2; }
  public int calculate() {
    int n1 = ept1.calculate(); int n2 = ept2.calculate();
    return n1 * n2; }  }
```

AddParseTree can also be used to write MulParaeTree; these are the only difference between them



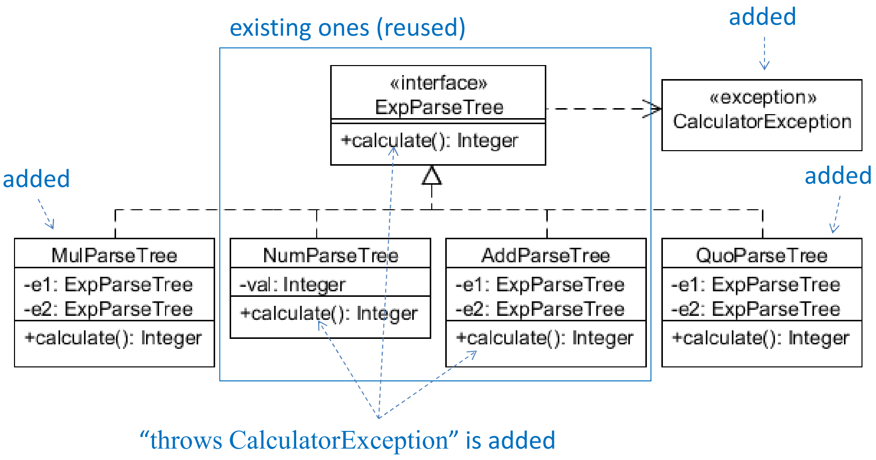
Parse tree for expression (5)

Division (quotient) makes us do some more because “divided by zero” exception may occur, but existing classes & interface can be essentially reused.
Necessary to add “throws CalculatorException” to the method calculate() in ExpParseTree & all existing classes that implements ExpParseTree except for NumParseTree.

```
public class QuoParseTree implements ExpParseTree {
  private ExpParseTree ept1, ept2;
  public QuoParseTree(ExpParseTree e1, ExpParseTree e2) {
      ept1 = e1; ept2 = e2; }
  public int calculate() throws CalculatorException {
    int n1 = ept1.calculate(); int n2 = ept2.calculate();
    if (n2 == 0) { throw new CalculatorException(); }
    return n1 / n2; }  }
```

an exception is thrown

Parse tree for expression (6)



existing ones (reused)

added

added

added

«interface»
ExpParseTree
+calculate(): Integer

«exception»
CalculatorException

MulParseTree
-e1: ExpParseTree
-e2: ExpParseTree
+calculate(): Integer

NumParseTree
-val: Integer
+calculate(): Integer

AddParseTree
-e1: ExpParseTree
-e2: ExpParseTree
+calculate(): Integer

QuoParseTree
-e1: ExpParseTree
-e2: ExpParseTree
+calculate(): Integer

“throws CalculatorException” is added

Parse tree for assignment programs (1)

Let us design parse trees for assignment programs; an assignment program is a sequence of assignment statements.
An example of assignment programs is as follows:

variable ---→

```
x := 2;
x := x*x;
x := x*x;
x := x*x;
```

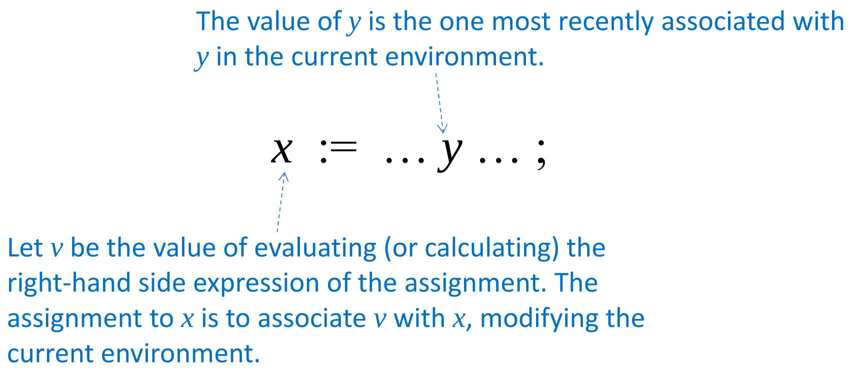
the program consists of four assignments

After the execution of the 4th assignment, x should be 256.

But, how to deal with variables?

Parse tree for assignment programs (2)

A map can be used to deal with variables.
Such a map is called an environment or a store.



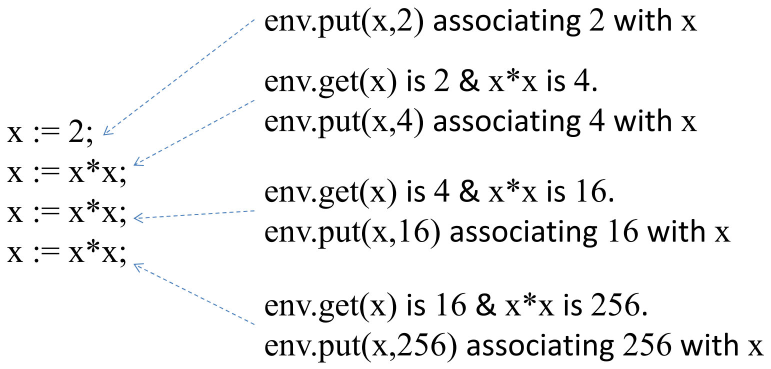
The value of y is the one most recently associated with y in the current environment.

$$x \;:=\; \ldots y \ldots ;$$

Let v be the value of evaluating (or calculating) the right-hand side expression of the assignment. The assignment to x is to associate v with x, modifying the current environment.

Parse tree for assignment programs (3)

Let env be the empty map.



```
x := 2;
x := x*x;
x := x*x;
x := x*x;
```

env.put(x,2) associating 2 with x

env.get(x) is 2 & x*x is 4.
env.put(x,4) associating 4 with x

env.get(x) is 4 & x*x is 16.
env.put(x,16) associating 16 with x

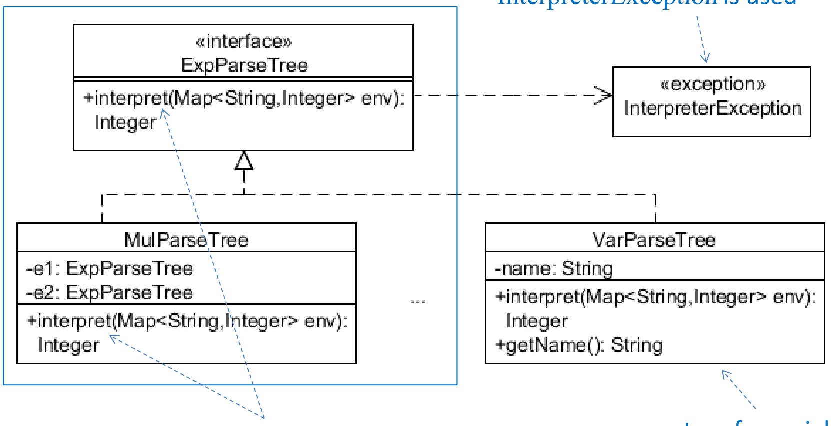
env.get(x) is 16 & x*x is 256.
env.put(x,256) associating 256 with x

After the execution of the 4th assignment, env.get(x) is 256.

Parse tree for assignment programs (4)

existing ones (reused)

Instead of CalculatorException, InterpreterException is used



interpret(…) is the same as calculate() except that the former has a map env as its parameter

parse tree for variables appearing in expressions

Parse tree for assignment programs (5)

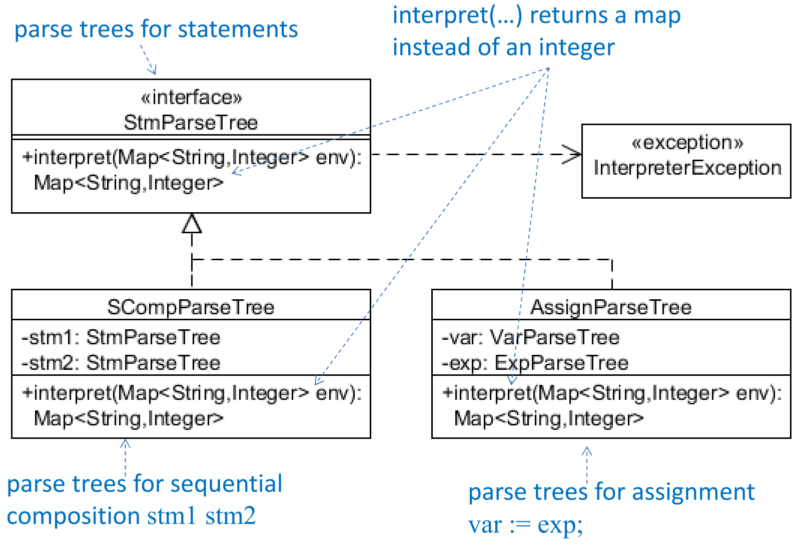
```
public class VarParseTree implements ExpParseTree {
  private String name;
  public VarParseTree(String s) { name = s; }
  public String getName() { return name; }

  public int interpret(Map<String,Integer> env)
        throws InterpreterException {
    try { return env.get(name); }
    catch (NullPointerException e) {
      throw new InterpreterException(name); } }  }
```

the name of the variable

the value of the variable

env.get(name) throws an exception NullPointerException if no value is associated with name; such an exception is caught here and then another exception InterpreterException is thrown

Parse tree for assignment programs (6)



parse trees for statements

interpret(…) returns a map instead of an integer

«interface»
StmParseTree
+interpret(Map<String,Integer> env):
Map<String,Integer>

«exception»
InterpreterException

SCompParseTree
-stm1: StmParseTree
-stm2: StmParseTree
+interpret(Map<String,Integer> env):
Map<String,Integer>

AssignParseTree
-var: VarParseTree
-exp: ExpParseTree
+interpret(Map<String,Integer> env):
Map<String,Integer>

parse trees for sequential composition stm1 stm2

parse trees for assignment var := exp;

Parse tree for assignment programs (7)

```
public interface StmParseTree {
  public Map<String,Integer> interpret(Map<String,Integer> env)
    throws InterpreterException;  }
                                    var := exp;
public class AssignParseTree implements StmParseTree {
  private VarParseTree var;  private ExpParseTree exp;
  public AssignParseTree(VarParseTree v, ExpParseTree e) {
    var = v;  exp = e;  }
  public Map<String,Integer> interpret(Map<String,Integer> env)
      throws InterpreterException {
    int n = exp.interpret(env); <------ evaluating  the expression exp
    env.put(var.getName(), n); <------ associating n with the variable var
    return env;  }  }
```

Parse tree for assignment programs (8)

stm1 stm2

```
public class SCompParseTree implements StmParseTree {
  private StmParseTree stm1, stm2;
  public SCompParseTree(StmParseTree s1, StmParseTree s2) {
    stm1 = s1;  stm2 = s2;  }
  public Map<String,Integer> interpret(Map<String,Integer> env)
      throws InterpreterException {
    Map<String,Integer> env1 = stm1.interpret(env);
    Map<String,Integer> env2 = stm2.interpret(env1);
    return env2;  }
```

executing (interpreting) stm1 using env, which may modify env; let env1 be the new environment

then, executing (interpreting) stm2 using env1, which may modify env1; let env2 be the new environment

finally, env2 is returned

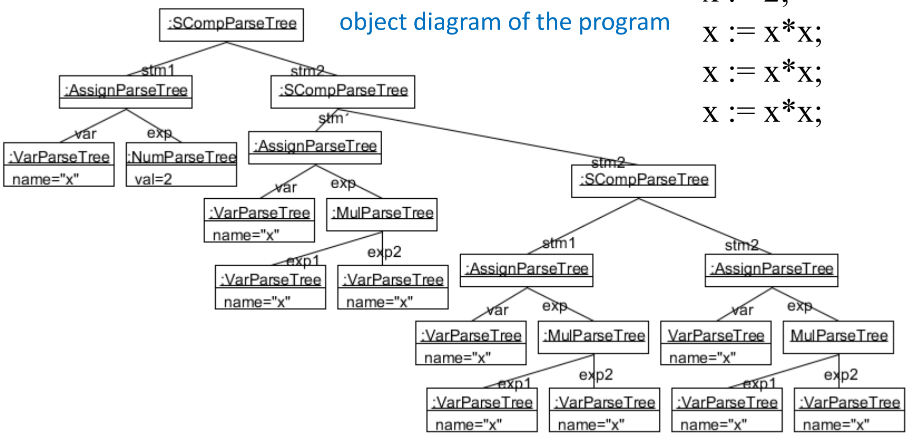
Parse tree for assignment programs (9)

```
Map<String,Integer> env = new HashMap<String,Integer>();
ExpParseTree two = new NumParseTree(2);
VarParseTree x = new VarParseTree("x");
ExpParseTree e = new MulParseTree(x,x);
StmParseTree a0 = new AssignParseTree(x,two);
StmParseTree a = new AssignParseTree(x,e);
StmParseTree s = new SCompParseTree(a,a);
s = new SCompParseTree(a,s);
s = new SCompParseTree(a0,s);
System.out.println(s.interpret(env));
```

the parse tree for the program

```
x := 2;
x := x*x;
x := x*x;
x := x*x;
```

Parse tree for assignment programs (10)



object diagram of the program

```
x := 2;
x := x*x;
x := x*x;
x := x*x;
```

Summary

- Information hiding
- Reuse
- Some examples for information hiding & reuse
  - Map
  - Parse tree for expression
  - Parse tree for assignment programs