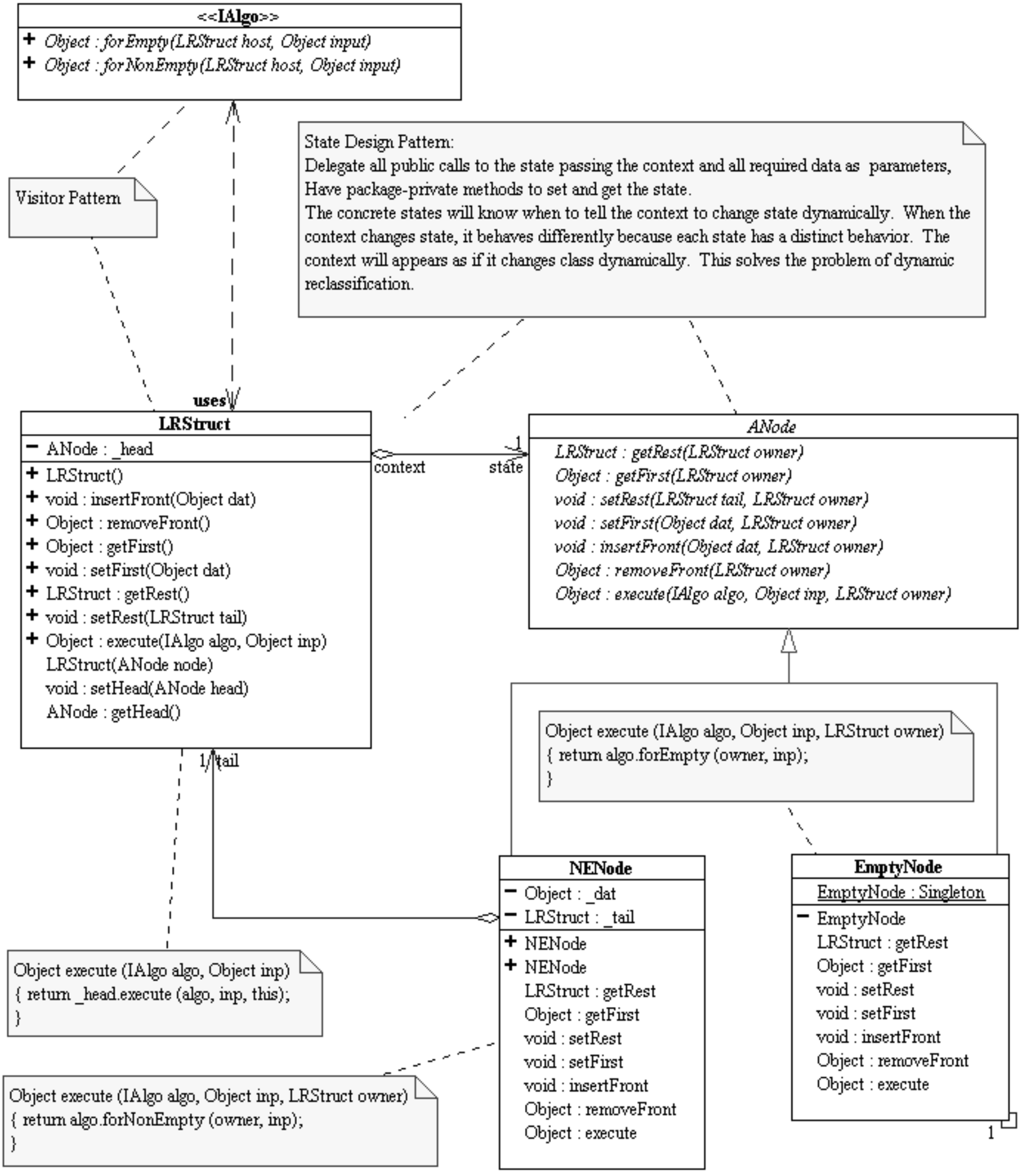
<<IAlgo>>
+ Object : forEmpty(LRStruct host, Object input)
+ Object : forNonEmpty(LRStruct host, Object input)
Visitor Pattern
State Design Pattern:
Delegate all public calls to the state passing the context and all required data as parameters,
Have package-private methods to set and get the state.
The concrete states will know when to tell the context to change state dynamically. When the context changes state, it behaves differently because each state has a distinct behavior. The context will appears as if it changes class dynamically. This solves the problem of dynamic reclassification.
uses
LRStruct
- ANode : _head
+ LRStruct()
+ void : insertFront(Object dat)
+ Object : removeFront()
+ Object : getFirst()
+ void : setFirst(Object dat)
+ LRStruct : getRest()
+ void : setRest(LRStruct tail)
+ Object : execute(IAlgo algo, Object inp)
LRStruct(ANode node)
void : setHead(ANode head)
ANode : getHead()
context
1
state
ANode
LRStruct : getRest(LRStruct owner)
Object : getFirst(LRStruct owner)
void : setRest(LRStruct tail, LRStruct owner)
void : setFirst(Object dat, LRStruct owner)
void : insertFront(Object dat, LRStruct owner)
Object : removeFront(LRStruct owner)
Object : execute(IAlgo algo, Object inp, LRStruct owner)
1
tail
Object execute (IAlgo algo, Object inp, LRStruct owner)
{ return algo.forEmpty (owner, inp);
}
NENode
- Object : _dat
- LRStruct : _tail
+ NENode
+ NENode
LRStruct : getRest
Object : getFirst
void : setRest
void : setFirst
void : insertFront
Object : removeFront
Object : execute
EmptyNode
EmptyNode : Singleton
- EmptyNode
LRStruct : getRest
Object : getFirst
void : setRest
void : setFirst
void : insertFront
Object : removeFront
Object : execute
1
Object execute (IAlgo algo, Object inp)
{ return _head.execute (algo, inp, this);
}
Object execute (IAlgo algo, Object inp, LRStruct owner)
{ return algo.forNonEmpty (owner, inp);
}