Layered Class Diagram

--- The way to express class structures of MixJuice programs ---

Yuuji Ichisugi

y-ichisugi@aist.go.jp

http://staff.aist.go.jp/y-ichisugi/mj/

National Institute of Advanced Industrial Science and Technology (AIST)

2002-10-30

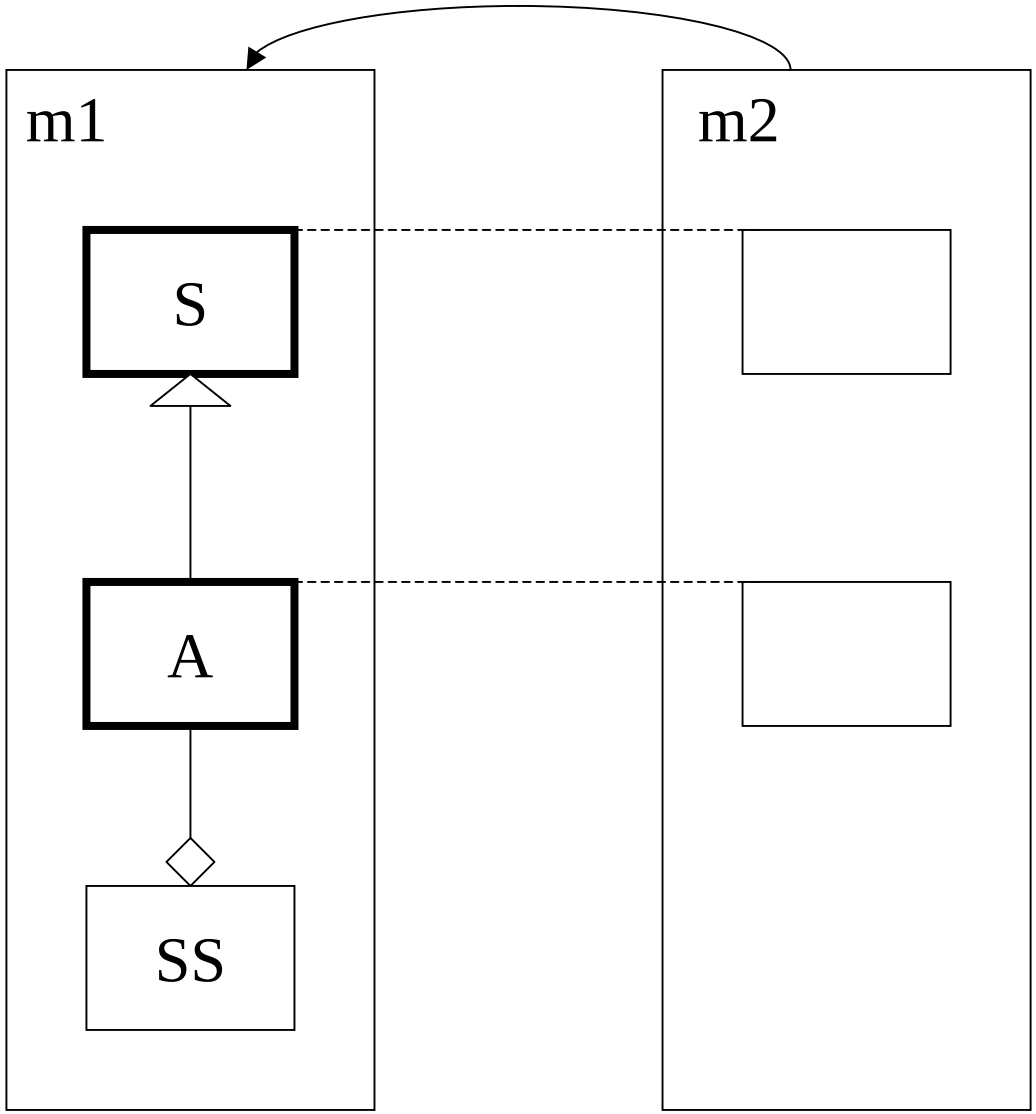
What is layered class diagram?

- Express how programs are extended by difference-based modules.
- Extension of ordinary class diagram.
  - All things expressed by class diagram can be expressed by layered class diagram.
- May be automatically generated from source code of MixJuice programs.

Sample program 1

```
module m1 {
  define class S {
    define int foo(){ return 1; }
  }
  define class A extends S {
    int foo(){ return original() + 10; }
  }
  class SS {
    void main(String[] args){ A a = new A(); … }
  }
}
```

```
module m2 extends m1 {
  class S { int foo(){ return original() + 2; } }
  class A { int foo(){ return original() + 20; }}
}
```

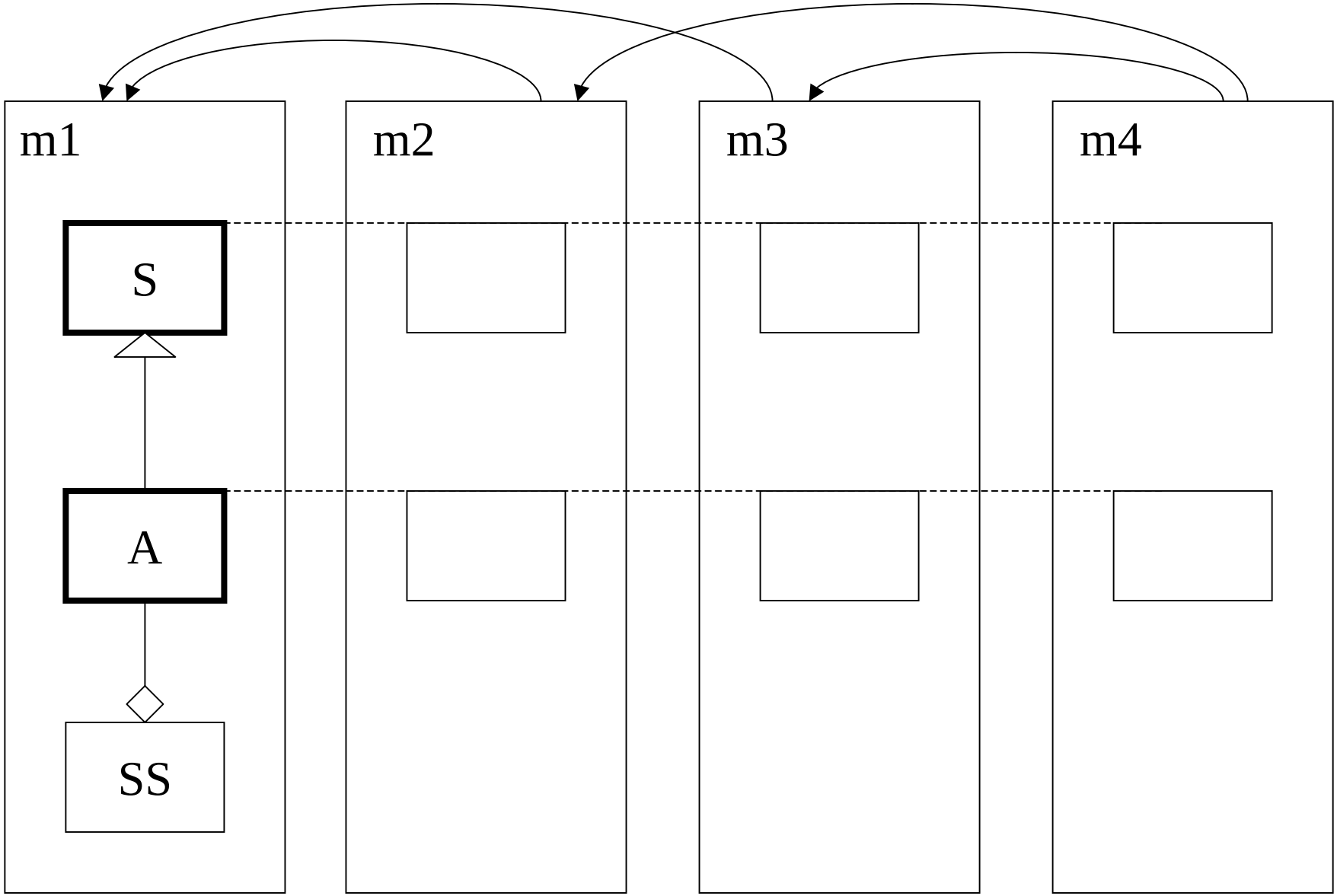

m1
m2
S
A
SS

Sample program 2

```
module m3 extends m1 {
  class S { int foo(){ return original() + 3; } }
  class A { int foo(){ return original() + 30; }}
}
```

```
module m4 extends m2,m3 {
  class S { int foo(){ return original() + 4; } }
  class A { int foo(){ return original() + 40; }}
}
```

Diamond inheritance of modules


m1
m2
m3
m4
S
A
SS

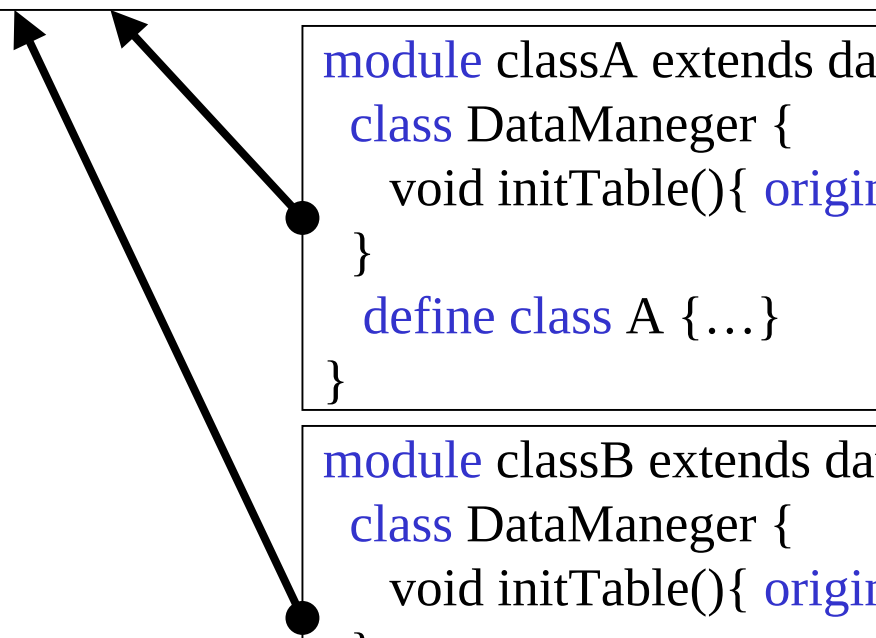
Sample program 3

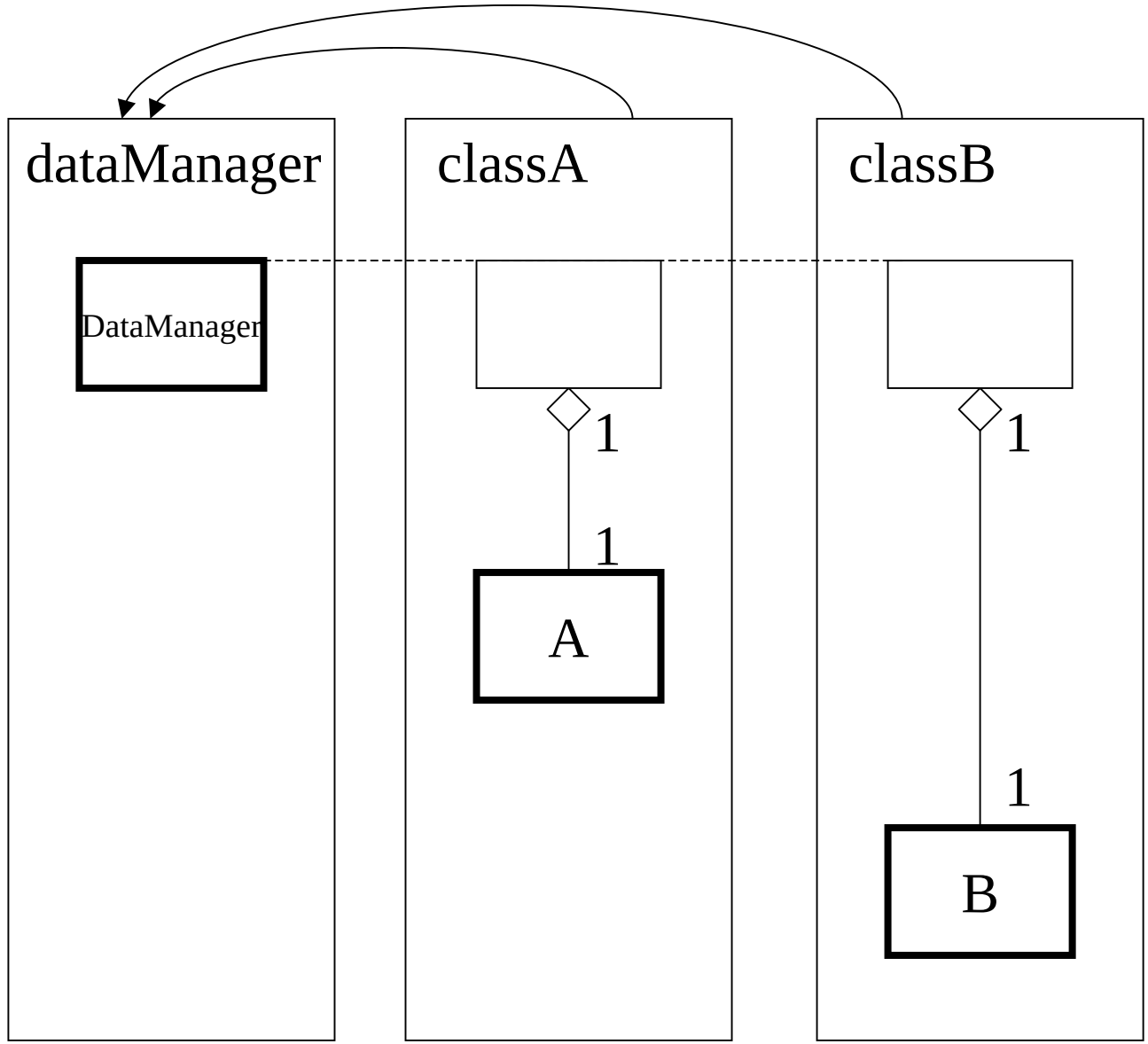
```
module dataManager {
  define class DataManeger {
     Hashtable table = new Hashtable();
     define void initTable(){}
  }
}
```

Framework which has an extensible method for table initialization

```
module classA extends dataManager {
  class DataManeger {
     void initTable(){ original(); table.put("A", new A()); }
  }
   define class A {…}
}
```

```
module classB extends dataManager {
  class DataManeger {
     void initTable(){ original(); table.put("B", new B()); }
  }
   define class B {…}
}
```




dataManager
classA
classB
DataManager
1
1
A
1
1
B

Sample program 4

```
module base {
  define class DrawFrame {
    Figure[] allFigures = null;
    …
  }
  define class Figure { … }
}
```

Extensible drawing tool framework

New figure

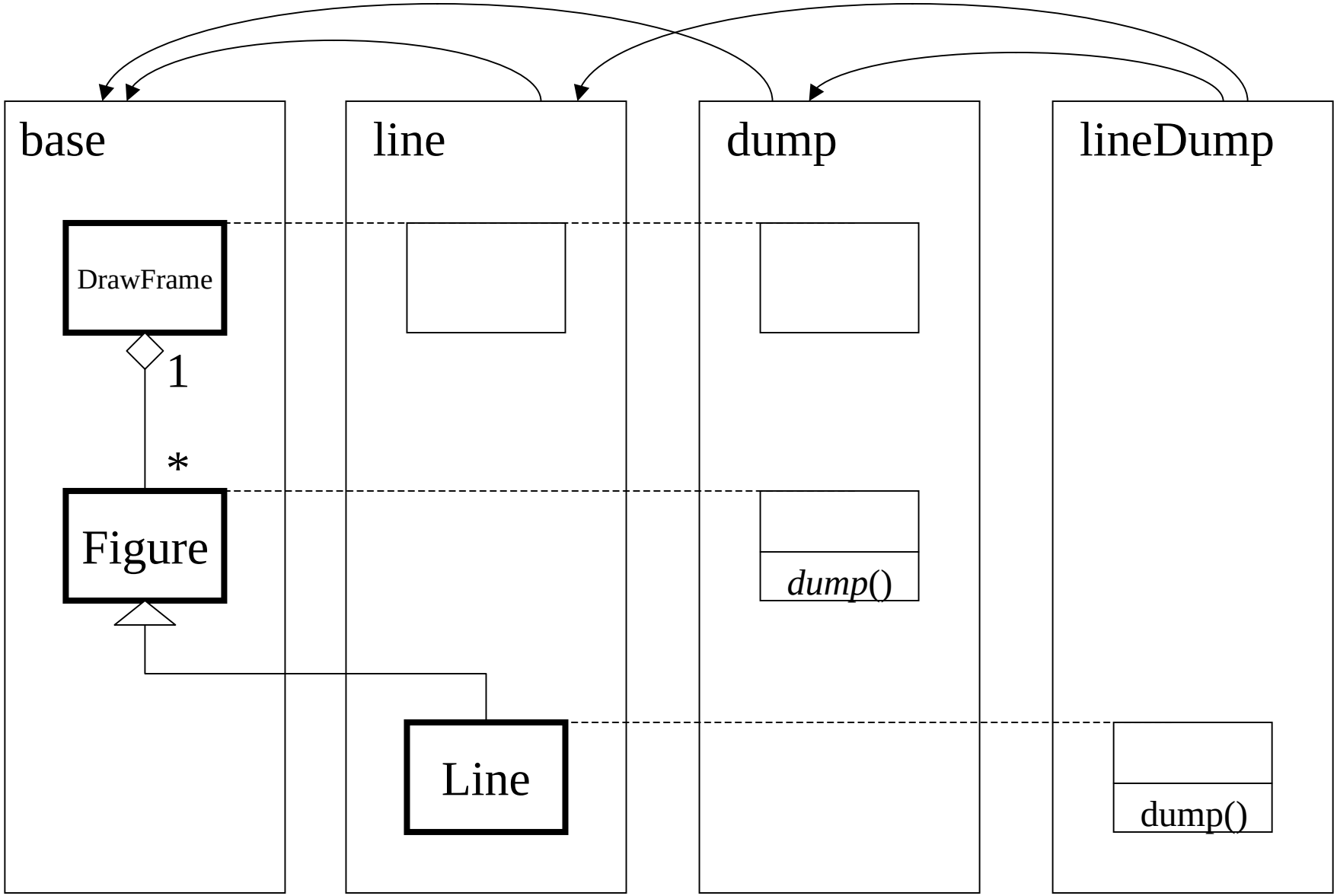
```
module line extends base {
 class DrawFrame { … }
 define class Line extends Figure { … }
}
```

New operation

```
module dump extends base {
  class DrawFrame { … }
  class Figure {
    define abstract void dump(); } }
```

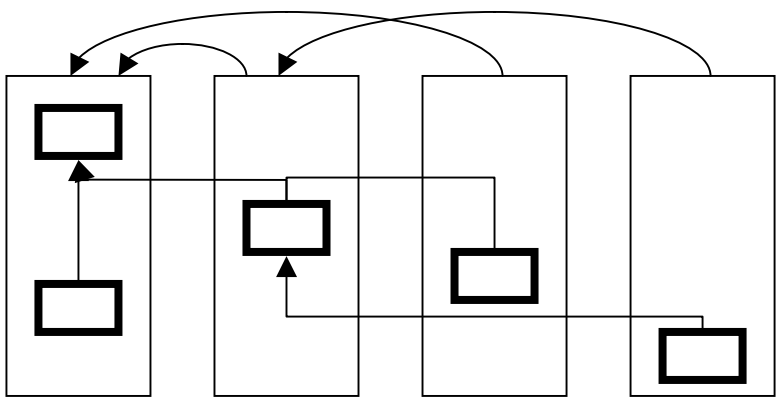
```
module lineDump complements line, dump {
  class Line { void dump(){ …} }
}
```

Complementary module


base
line
dump
lineDump
DrawFrame
1
*
Figure
dump()
Line
dump()

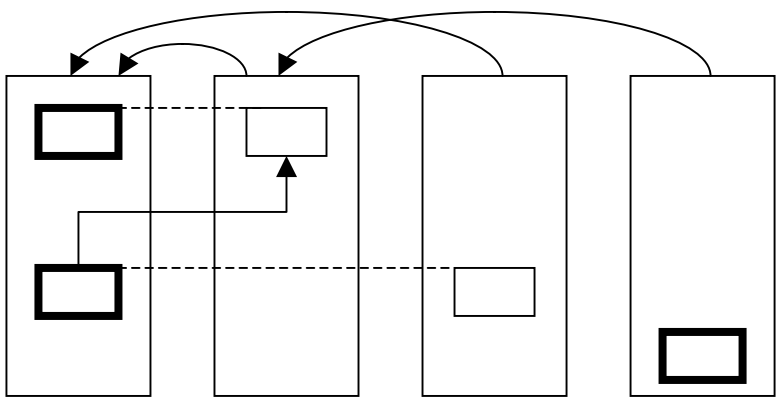
Writing rules (1/3)

- Vertical dimension is for classes, horizontal dimension is for modules.
- Classes should be sorted from top to down according to the class inheritance relation.
- Modules should be sorted from left to right according to the module inheritance relation.



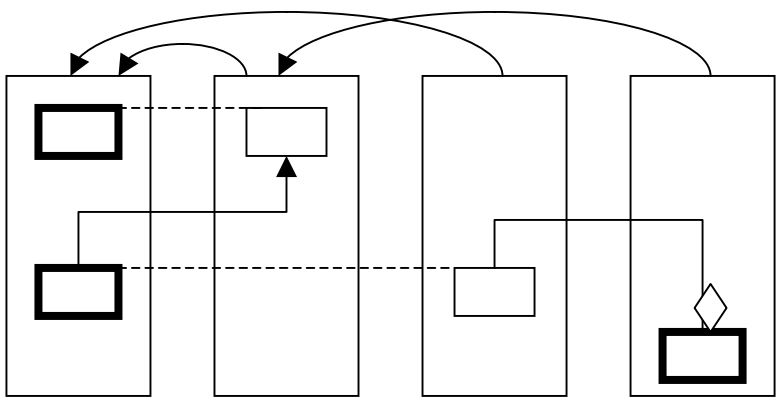
Writing rules (2/3)

- Class definitions are written with bold lines.
- The class definition and extensions for the same class should be located in the same vertical position and linked by dotted lines.

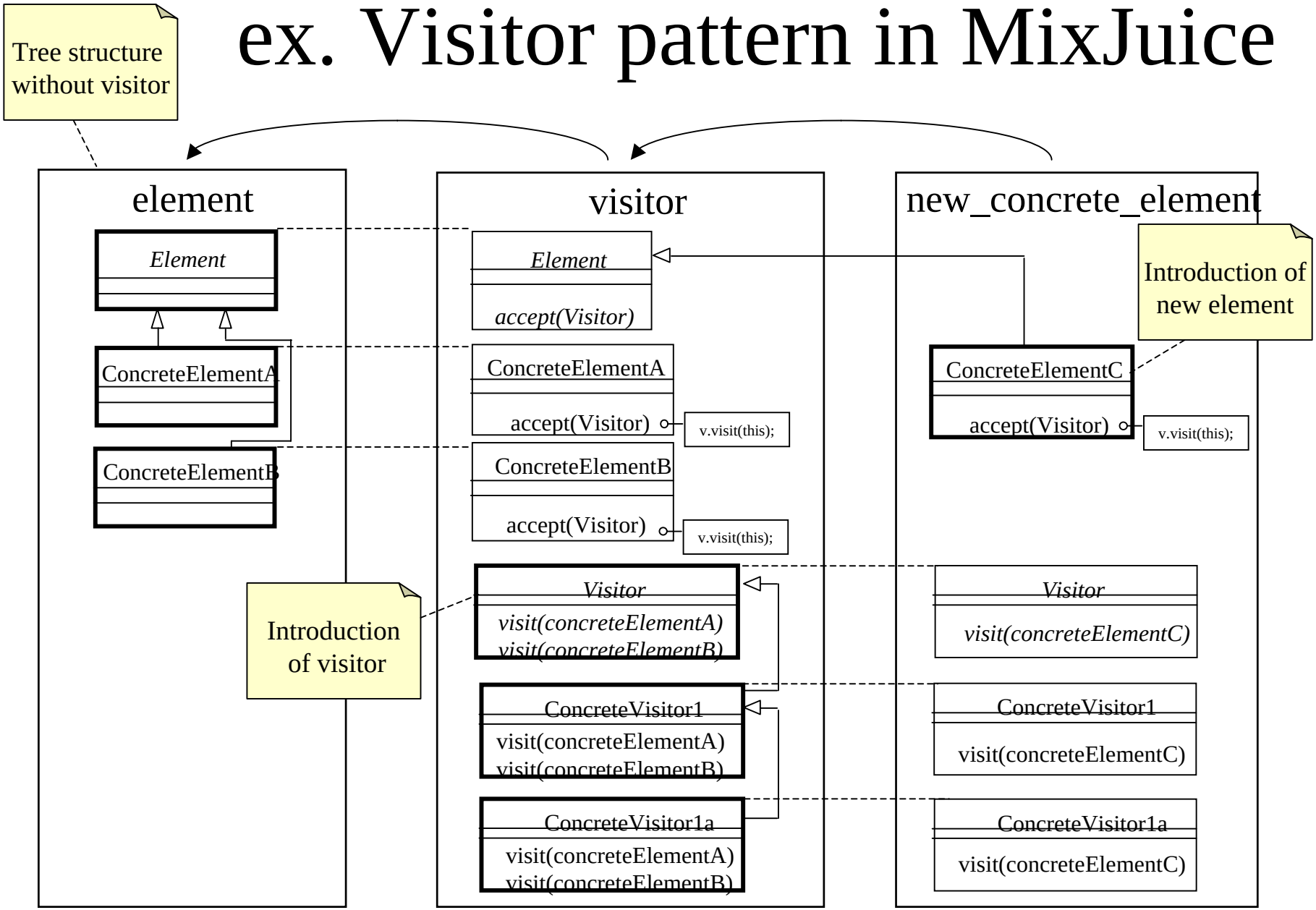


Writing rules (3/3)

- Each line for inheritance, aggregation etc. starts from the class fragment where it is introduced and ends at arbitrary fragment of the target class.



ex. Visitor pattern in MixJuice



Conclusion

- Layered class diagram is useful to explain how modules extend programs.
- Not only for MixJuice but also can be used for other AOP languages and open-class languages.
  - AspectJ, Hyper/J, mixin layers, PCA,…
  - CLOS, Objective-C, Cecil, Ruby, …
- **Comments are welcome!**