EECS3311 Software Design (Fall 2020)

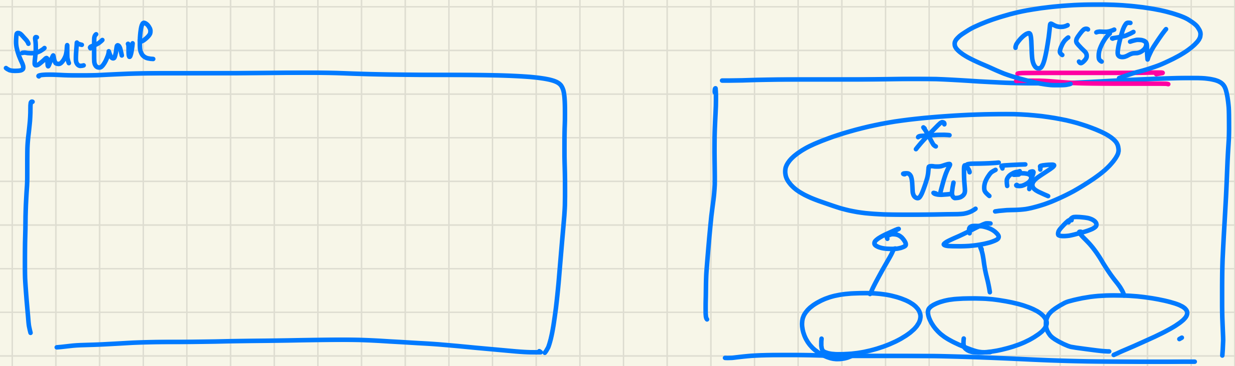
Q&A - Lecture Series W11

Monday, November 30

The name of visitor comes from the fact that we want to traverse through all the items in a data structure?

recursive

Can we review this pattern again because I cannot distinguish it from composite pattern?

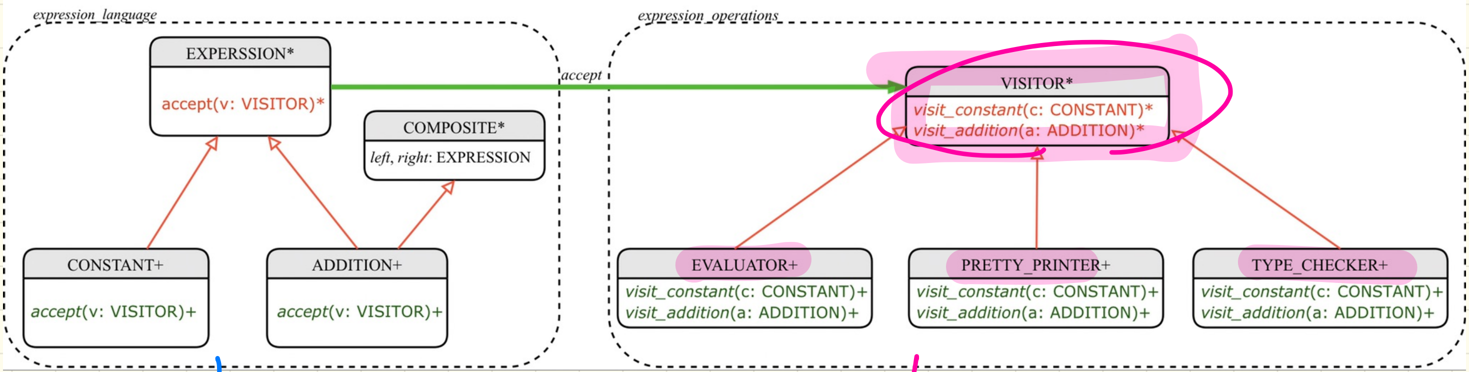


Use visitor if

① already composite implemented (recursive)

②. multiple operations to support

Visitor Design Pattern: Architecture



expression_language
EXPERSSION*
accept(v: VISITOR)*
COMPOSITE*
left, right: EXPRESSION
CONSTANT+
accept(v: VISITOR)+
ADDITION+
accept(v: VISITOR)+
accept
expression_operations
VISITOR*
visit_constant(c: CONSTANT)*
visit_addition(a: ADDITION)*
EVALUATOR+
visit_constant(c: CONSTANT)+
visit_addition(a: ADDITION)+
PRETTY_PRINTER+
visit_constant(c: CONSTANT)+
visit_addition(a: ADDITION)+
TYPE_CHECKER+
visit_constant(c: CONSTANT)+
visit_addition(a: ADDITION)+

↳ Composite (ability to (build recursive objects)

↳

How to Use Visitors

```
test_expression_evaluation: BOOLEAN
  local add, c1, c2: EXPRESSION ; v: VISITOR
  do
    create {CONSTANT} c1.make (1) ; create {CONSTANT} c2.make (2)
    create {ADDITION} add.make (c1, c2)
    create {EVALUATOR} v.make
    add.accept(v)
    check attached {EVALUATOR} v as eval then
      Result := eval.value = 3
    end
  end
```

v.value ✗

→ using composite

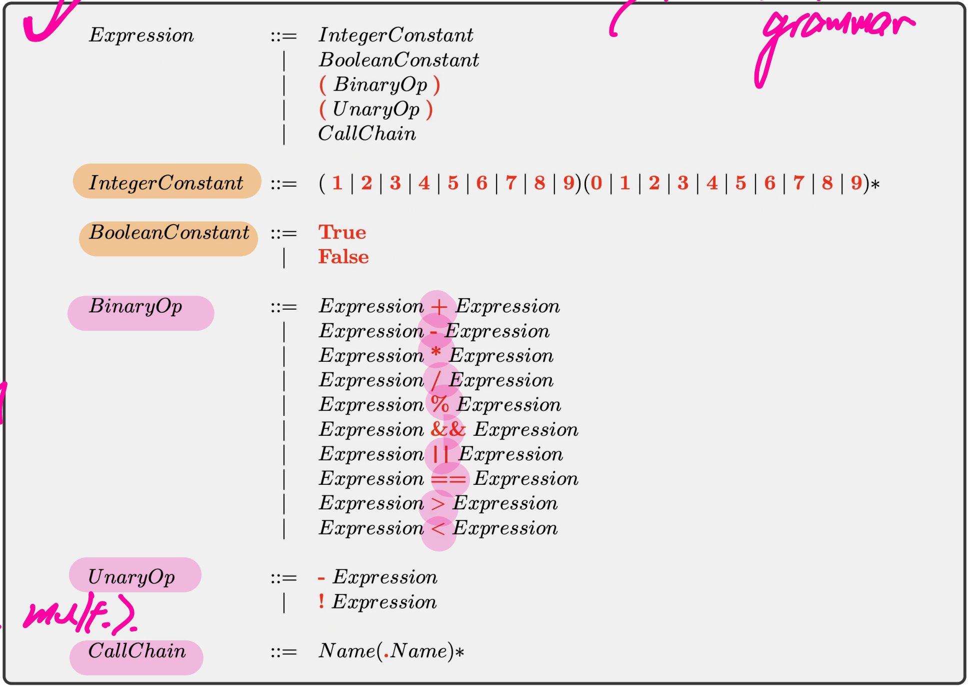
→ using visitor.

I would like to verify that if I want to implement a MULTIPLICATION class, I must be aware that I will need it ,
and I should have implemented it in the beginning along with the ADD class instead of adding the MULTIPLICATION class in after everything is implemented.
Is that correct?

✓

① the "structures" cluster in the first working version of the system should contain as many classes as you can anticipate.

② Otherwise, each addition of a new structure class (e.g. mult.) will violate SCP.



```
Expression       ::= IntegerConstant
                   | BooleanConstant
                   | ( BinaryOp )
                   | ( UnaryOp )
                   | CallChain

IntegerConstant  ::= ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)(0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)*

BooleanConstant  ::= True
                   | False

BinaryOp         ::= Expression + Expression
                   | Expression - Expression
                   | Expression * Expression
                   | Expression / Expression
                   | Expression % Expression
                   | Expression && Expression
                   | Expression || Expression
                   | Expression == Expression
                   | Expression > Expression
                   | Expression < Expression

UnaryOp          ::= - Expression
                   | ! Expression

CallChain        ::= Name(.Name)*
```

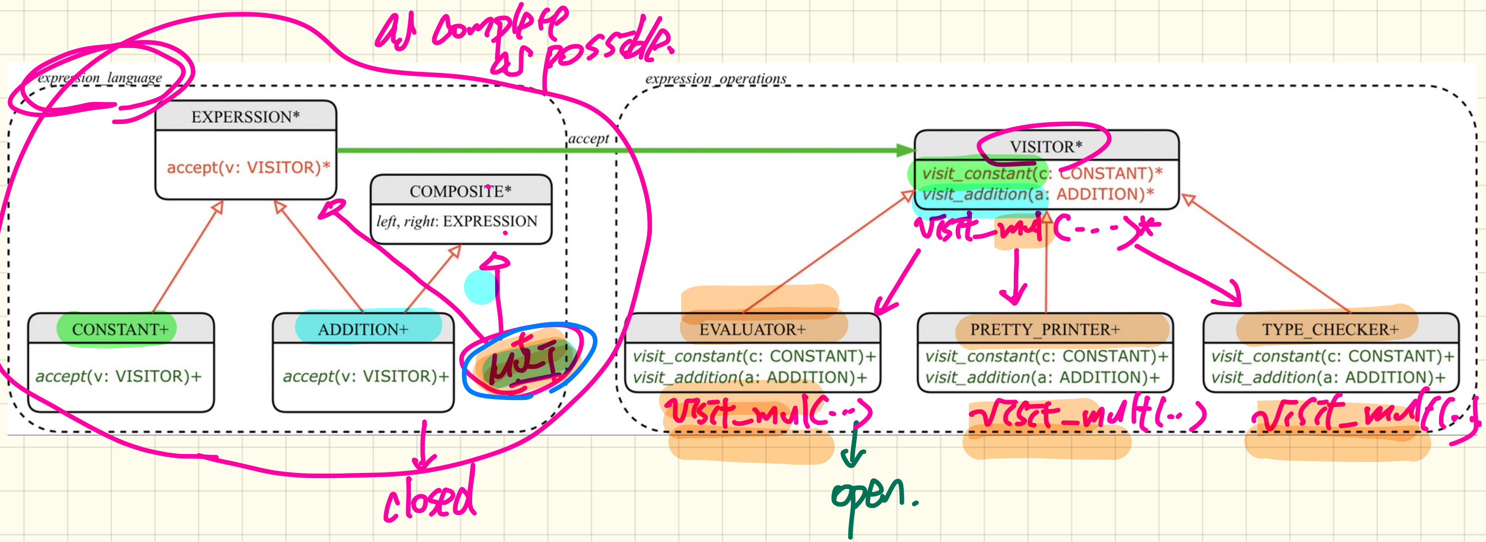
context-free grammar

there is a correlation between classes in expression cluster (constant, addition) to the commands of expression operations cluster (visit_constant, visit_addition) while the expression language is closed and we are not going to make changes, then being open in the expression operation cluster means what?

↳ ① add as many descendant classes of VISITOR as you like

② Each new descendant class you add, that's the single point of changes.

Visitor Pattern: Open-Closed and Single-Choice Principles

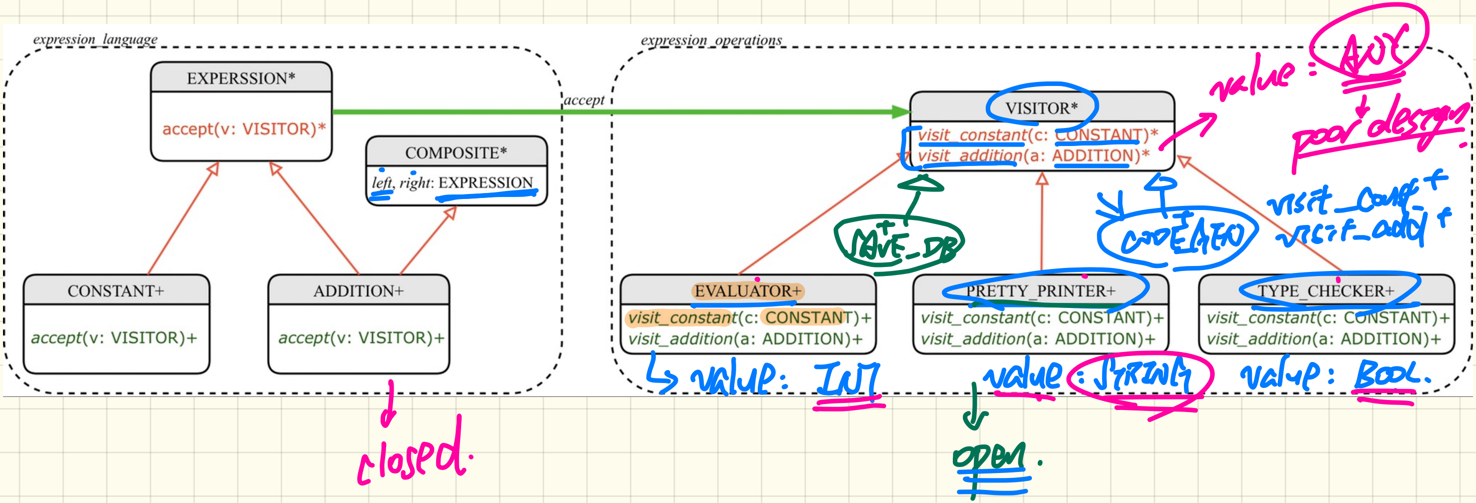


What if a new language construct is added?

↳ violates SCP.

If the visitor pattern is adopted, what should be closed?

Visitor Pattern: Open-Closed and Single-Choice Principles



What if a new language operation is added?

↳ SCP satisfied

If the visitor pattern is adopted, what should be open?

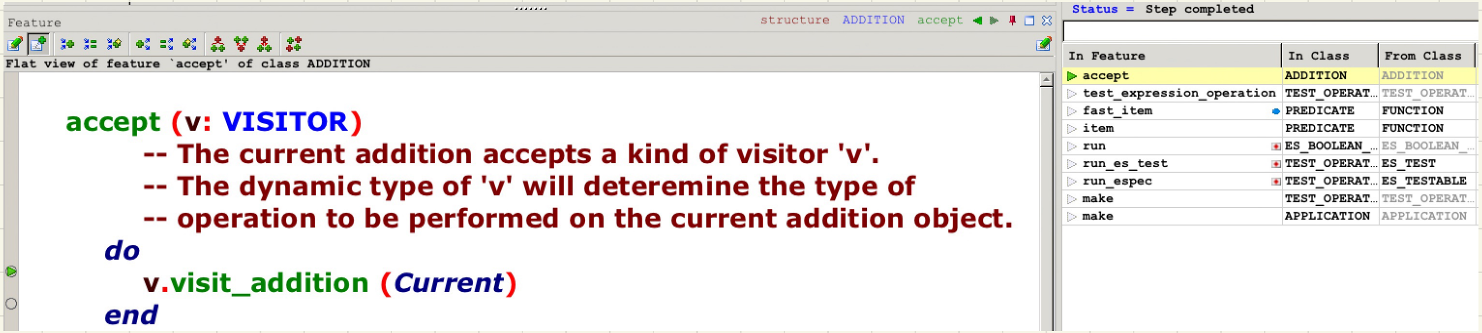
From Composite-Visitor Totorial:

I can tell the static type of v is VISITOR by "(v: VISITOR)", but why is its dynamic type is EVALUATOR?

I couldn't find a line code to assign that. create eval.make

✓ Is it because EVALUATOR is the only class that has the "visit_constant" effective, so EVALUATOR automatically becomes the default type when "visit_constant" is called. X

What if we have the PRETTY_PRINTER class, how can the second dispatch decide which one will it call the "visit_constant".



Flat view of feature `accept' of class ADDITION

```
accept (v: VISITOR)
        -- The current addition accepts a kind of visitor 'v'.
        -- The dynamic type of 'v' will deteremine the type of
        -- operation to be performed on the current addition object.
    do
        v.visit_addition (Current)
    end
```

Status = Step completed

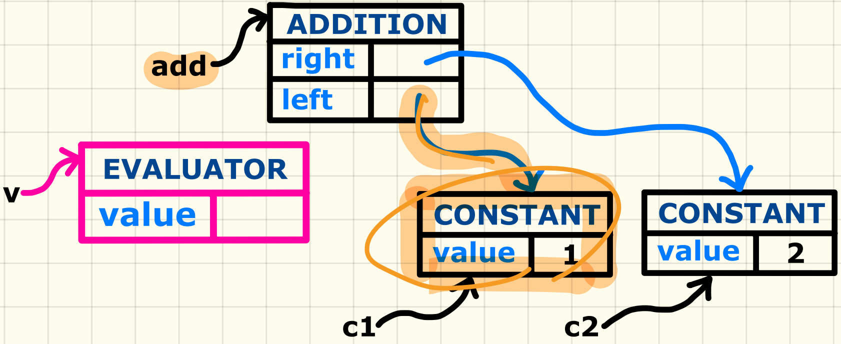
In Feature	In Class	From Class
accept	ADDITION	ADDITION
test_expression_operation	TEST_OPERAT...	TEST_OPERAT...
fast_item	PREDICATE	FUNCTION
item	PREDICATE	FUNCTION
run	ES_BOOLEAN_...	ES_BOOLEAN_...
run_es_test	TEST_OPERAT...	ES_TEST
run_espec	TEST_OPERAT...	ES_TESTABLE
make	TEST_OPERAT...	TEST_OPERAT...
make	APPLICATION	APPLICATION

Executing **Composite** and **Visitor** Patterns at **Runtime**

1st D.S.

Tracing add.accept(v)

Double Dispatch



exp. accept (visitor)

↳ triggers a D.S.

```
deferred class VISITOR
  visit_constant(c: CONSTANT) deferred end
  visit_addition(a: ADDITION) deferred end
end
```

```
class CONSTANT inherit EXPRESSION
...
  accept(v: VISITOR)
    do
      v.visit_constant (Current)
    end
end
```

eval_left

DT: EVALUATOR

Alt. v.visit_addition(Curr.)

```
class EVALUATOR inherit VISITOR
  value : INTEGER
  visit_constant(c: CONSTANT) do value := c.value end
  visit_addition(a: ADDITION)
    local eval_left, eval_right: EVALUATOR
    do a.left.accept(eval_left)
       a.right.accept(eval_right)
       value := eval_left.value + eval_right.value
    end
end
```

2nd D.S.

visitor

Exp.

DT: CONSTANT

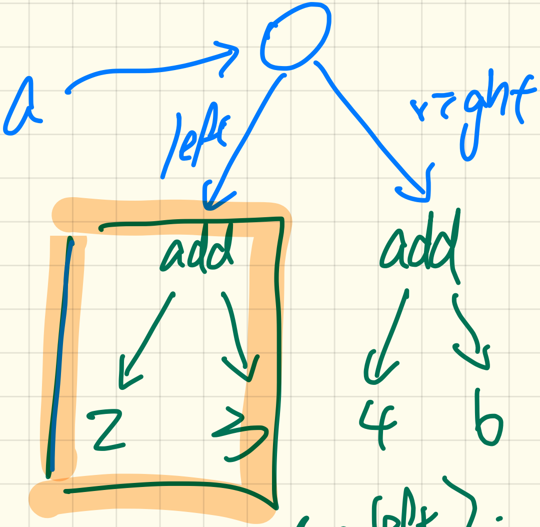
2nd D.S.

base case

no further D.S.

```
class ADDITION
inherit EXPRESSION COMPOSITE
...
  accept(v: VISITOR)
    do
      v.visit_addition(Current)
    end
end
```

Compilation error.



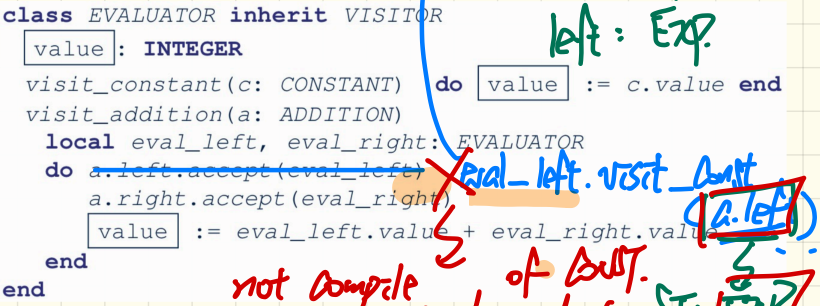
$\underbrace{2}_{left} + \underbrace{3}_{right}$

$\underbrace{(2+3)}_{a.left} + \underbrace{(4+6)}_{a.right}$

→ v. visit_Constant (a.left).

VISITOR:
visit_const. (c: CONST.)

add_EXP:
left: EXP



```
class EVALUATOR inherit VISITOR
  value : INTEGER
  visit_constant(c: CONSTANT) do value := c.value end
  visit_addition(a: ADDITION)
    local eval_left, eval_right: EVALUATOR
    do ~~a.left.accept(eval_left)~~
      a.right.accept(eval_right)
      value := eval_left.value + eval_right.value
    end
end
```

eval_left.visit_Const (a.left)

not compile

of CONST.

✓ EXP not descendant

ST: EXP

I think open/close concept is not.. practical to follow.

As an example, in the project (SD2), I am always changing all the parts of all the objects even the very first object I created , because otherwise I cannot satisfy the requirements in a proper way.

in the course of implementing a working version, not yet subject to OCP.

How do I decide where to draw the line between open and close ? What am I missing? On a related question, in a team work project, how do we follow OCP?

Open or Closed

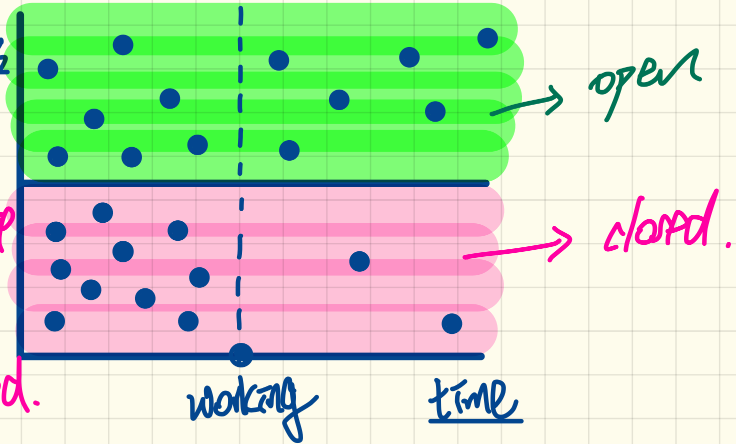
orb b. s. g. changes

orbment.

focus!

enemy type

open if you anticipate more enemy types to be added.



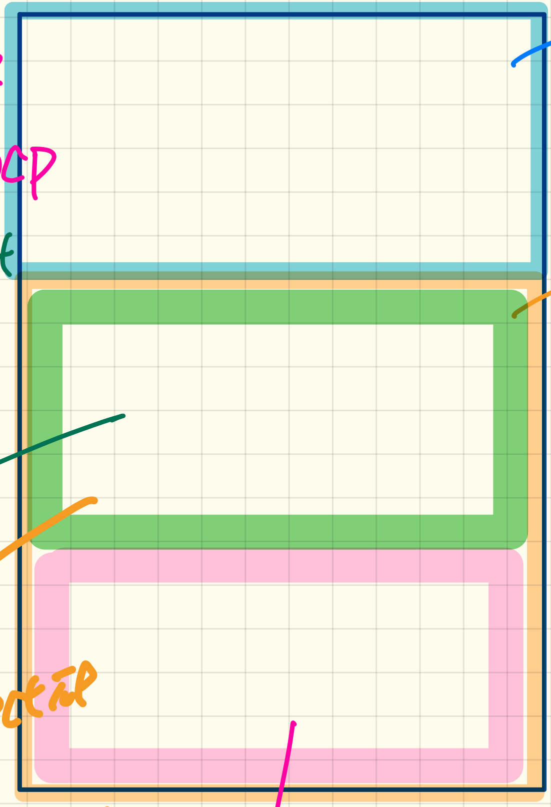
① Which part → OCP

Which part $\overset{\times}{\rightarrow}$ OCP.

② For the part → OCP

↳ Which sub-part open?

↳ Which sub-part closed?



part of system not subject to OCP.

In practice, choose only a part of your system that's subject to the OCP.

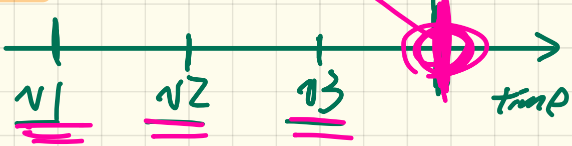
open

OCP starts being effective after some working version is done.

closed.

point where design principles should be satisfied

submission deadline


v1
v2
v3
time