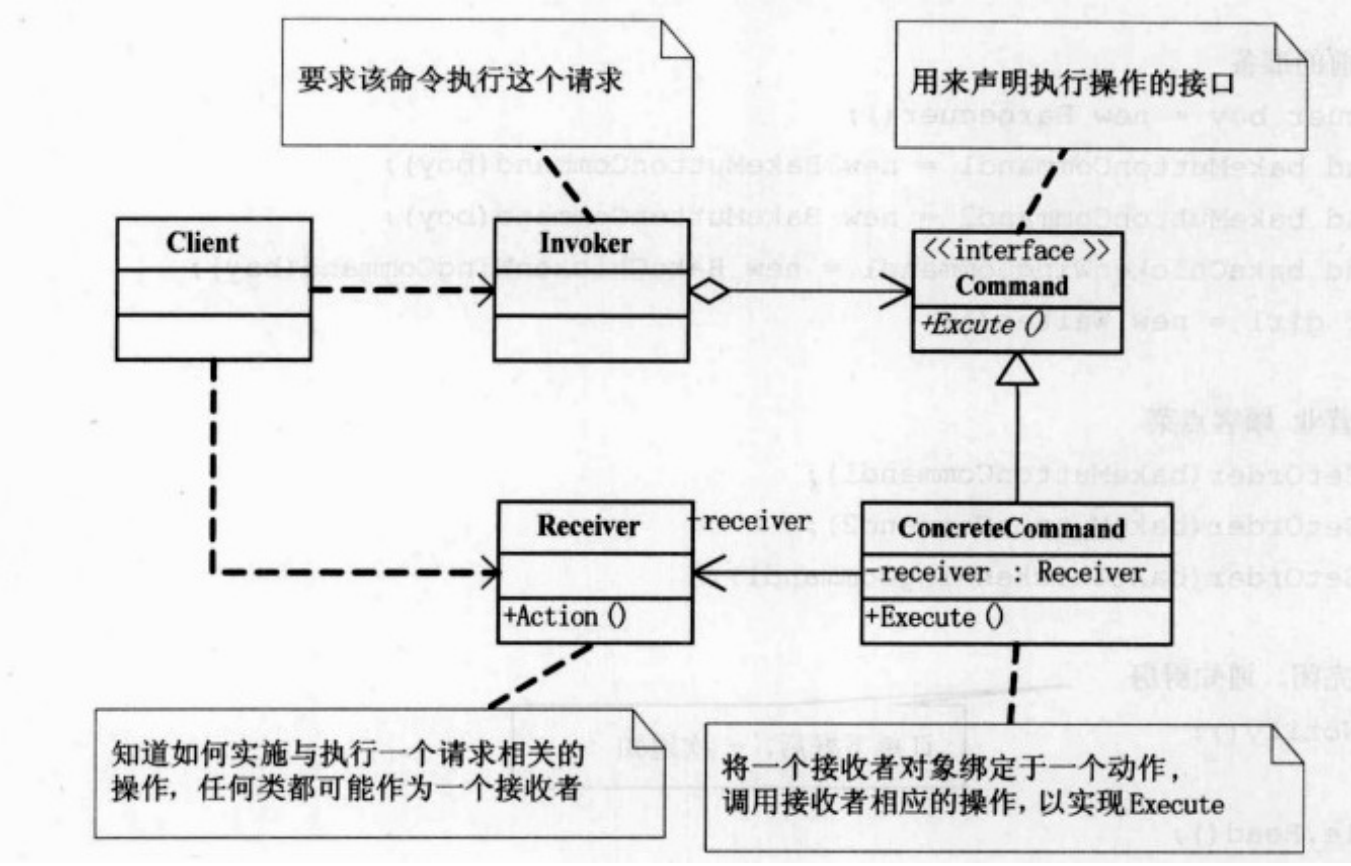
SAP Interview

1. Why SAP

As a new graduate, I am alwasy fascinated by new technoloy. Working at SAP and developing next generation cloud-based analytics platform attracts me a lot. Also, I have heard many great things about SAP from my friend Yang. I know SAP has many talented engineers and it has friendly work culture and policies. Therefore, SAP is a place that I can learn a lot and make things happen with my team.

2. Paint




```
class CommandManager(object):
    def __init__(self):
        self.undo_commands = []
        self.redo_commands = []

    def push_undo_command(self, command):
```

```
7.          """Push the given command to the undo command stack."""
8.          self.undo_commands.append(command)
9.
10.     def pop_undo_command(self):
11.         """Remove the last command from the undo command stack and
    return it.
12.         If the command stack is empty, EmptyCommandStackError is
    raised.
13.
14.         """
15.         try:
16.             last_undo_command = self.undo_commands.pop()
17.         except IndexError:
18.             raise EmptyCommandStackError()
19.         return last_undo_command
20.
21.     def push_redo_command(self, command):
22.         """Push the given command to the redo command stack."""
23.         self.redo_commands.append(command)
24.
25.     def pop_redo_command(self):
26.         """Remove the last command from the redo command stack and
    return it.
27.         If the command stack is empty, EmptyCommandStackError is
    raised.
28.
29.         """
30.         try:
31.             last_redo_command = self.redo_commands.pop()
32.         except IndexError:
33.             raise EmptyCommandStackError()
34.         return last_redo_command
35.
36.     def do(self, command):
37.         """Execute the given command. Exceptions raised from the
    command are
38.         not catched.
39.
40.         """
41.         command()
42.         self.push_undo_command(command)
43.         # clear the redo stack when a new command was executed
44.         self.redo_commands[:] = []
45.
46.     def undo(self, n=1):
```

```
        """Undo the last n commands. The default is to undo only the last
        command. If there is no command that can be undone because n is too big
        or because no command has been emitted yet, EmptyCommandStackError is
        raised.

        """
        for _ in xrange(n):
            command = self.pop_undo_command()
            command.undo()
            self.push_redo_command(command)

    def redo(self, n=1):
        """Redo the last n commands which have been undone using the undo
        method. The default is to redo only the last command which has been
        undone using the undo method. If there is no command that can be redone
        because n is too big or because no command has been undone yet,
        EmptyCommandStackError is raised.

        """
        for _ in xrange(n):
            command = self.pop_redo_command()
            command()
            self.push_undo_command(command)
```

3. design an vending machine (设计自动贩卖机)

```
class Item:
    def __init__(self, name, price, stock):
        self.name = name
        self.price = price
        self.stock = stock

    def updateStock(self, stock):
        self.stock = stock

    def buyFromStock(self):
```

```
        if self.stock == 0:
            # raise not item exception
            pass
        self.stock -= 1

class VendingMachine:
    def __init__(self):
        self.amount = 0
        self.items = []

    def addItem(self, item):
        self.items.append(item)

    def showItems(self):
        print('\nitems available \n***************')

        for item in self.items:
            if item.stock == 0:
                self.items.remove(item)
        for item in self.items:
            print(item.name, item.price)

        print('***************\n')

    def addCash(self, money):
        self.amount = self.amount + money

    def buyItem(self, item):
        if self.amount < item.price:
            print('You can\'t but this item. Insert more coins.')
        else:
            self.amount -= item.price
            item.buyFromStock()
            print('You got ' +item.name)
            print('Cash remaining: ' + str(self.amount))

    def containsItem(self, wanted):
        ret = False
        for item in self.items:
            if item.name == wanted:
                ret = True
                break
        return ret

    def getItem(self, wanted):
```

```
            ret = None
            for item in self.items:
                if item.name == wanted:
                    ret = item
                    break
            return ret

        def insertAmountForItem(self, item):
            price = item.price
            while self.amount < price:
                    self.amount = self.amount + float(input('insert ' + str
    (price - self.amount) + ': '))

        def checkRefund(self):
            if self.amount > 0:
                print(self.amount + " refunded.")
                self.amount = 0

            print('Thank you, have a nice day!\n')
```

3. Difference between primary and foreign keys

Primary Key: identify uniquely every row it can not be null. it can not be a duplicate.

Foreign Key: create relationship between two tables. can be null. can be a duplicate