1. 实体型对象

- 代码段

```python
#!/usr/bin/env python
# -*- encoding: utf-8 -*-
'''
@File    :   crational.py
@Time    :   2019/10/12 11:30:04
@Author  :   Qu Yuanbin
@Version :   1.0
@Contact :   2191002033@cnu.edu.cn
@License :
@Desc    :   实现分数计算
'''

# here put the import lib


class CRational:
    """ 分数类 """
    __slots__ = ('_n', '_d')

    def __init__(self, n=0, d=1):
        self._n = n  # 分子
        self._d = d  # 分母

    def __str__(self):
        return ('%d/%d' % (self._n, self._d))

    def _gcd(self, n, d):
        """ 求最大公约数 """
        for i in range(1, min(n, d)):
            if n % i == 0 and d % i == 0:
                result = i
        return int(result)

    def _lcm(self, n, d):
        """ 求最小公倍数 """
        return int(n * d / self._gcd(n, d))

    @property
    def n(self):
        return self._n

    @property
    def d(self):
        return self._d
```

```
    @n.setter
    def n(self, n):
        self._n = n

    @d.setter
    def d(self, d):
        self._d = d

    def process(self, r, operator):
        """ 运算 """
        result = CRational()
        if operator == '+':  # 加法
            result.d = self._lcm(self.d, r.d)
            result.n = (result.d // self.d * self.n) + (result.d
// r.d * r.n)
        elif operator == '-':  # 减法
            result.d = self._lcm(self.d, r.d)
            result.n = (result.d // self.d * self.n) - (result.d
// r.d * r.n)
        elif operator == '*':  # 乘法
            result.d = self.d * r.d
            result.n = self.n * r.n
        gcd = self._gcd(result.n, result.d)
        if gcd != 1:  # 化简结果
            result.n = (result.n // gcd)
            result.d = (result.d // gcd)
        return result

    def gt(self, r):
        """ 大于 """
        return True if(self.process(r, '-').n > 0) else False

    def lt(self, r):
        """ 小于 """
        return True if(self.process(r, '-').n < 0) else False

    def eq(self, r):
        """ 等于 """
        return True if(self.process(r, '-').n == 0) else False


def compare(op1, op2):
    """ 比较运算符优先级 """
    return op1 in ['*', '-'] and op2 in ['+', '-']


def process(data, opt):
    """ 运算 """
    num1 = data.pop()
    num2 = data.pop()
    operator = opt.pop()
    data.append(num2.process(num1, operator))
```

```
def get_crational(s, start, end):
    """ 构建一个分数 """
    n, d = s[start: end].split('/')
    return CRational(int(n), int(d))


def calculate(s):
    """ 计算字符串表达式的值 """
    data = []  # 数据栈
    opt = []  # 操作符栈
    i = 0  # 表达式遍历
    while i < len(s):
        if s[i].isdigit():  # 数字，向后匹配到整个分数
            start = i
            while i + 1 < len(s) and s[i + 1] not in ['+', '-',
'*']:
                i += 1
            data.append(get_crational(s, start, i + 1))
        elif s[i] == ')':  # 右括号
            while opt[-1] != "(":
                process(data, opt)
            opt.pop()  # 出栈"("
        elif not opt or opt[-1] == '(':  # 操作符栈为空或者栈顶元素为
左括号，则入栈
            opt.append(s[i])
        elif s[i] == '(' or compare(s[i], opt[-1]):  # 前操作符为左
括号或者比栈顶操作符优先级高，则入栈
            opt.append(s[i])
        else:
            while opt and not compare(s[i], opt[-1]):
                if opt[-1] == '(':  # 若遇到左括号，停止计算
                    break
                process(data, opt)
            opt.append(s[i])
        i += 1
    while opt:
        process(data, opt)
    print(data.pop())


if __name__ == '__main__':
    s = input('输入要计算的表达式:')
    calculate(s)
```

- 运行结果

```
~/Code_Learning(master*) » /home/qu/anaconda3/envs/pytorch/bin/python /home/qu/Code_Learning/Course/面向对象方法学/第三次作业/crational.py
输入要计算的表达式:13/24*4/7-3/29
251/1218
```

2.加工型对象

- 代码段

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-
'''
@File    :   get_longest.py
@Time    :   2019/10/12 14:39:15
@Author  :   Qu Yuanbin
@Version :   1.0
@Contact :   2191002033@cnu.edu.cn
@License :
@Desc    :   获取文本最长行及其长度
'''

# here put the import lib
import sys
import os


class CText:
    """ 文本类 """
    __slots__ = ('_in', '_out')

    def __init__(self, t_in, out):
        self._in = t_in
        self._out = out

    @property
    def out(self):
        return self._out

    def get_longest(self):
        """ 获取最长句子 """
        num_list = [len(line) for line in self._in]
        max_length = num_list[num_list.index(max(num_list))]
        max_indexs = [i for i, num in enumerate(num_list) if num
== max_length]
        for index in max_indexs:
            self._out.add(self._in[index], index)


class CBuf:
    """ 暂存类 """
    __slots__ = ('_num', '_bufs', '_nos')

    def __init__(self):
        self._num = 0
        self._bufs = []
        self._nos = []
```

```
    def clear(self):
        """ 清空 """
        self._num = 0
        self._bufs.clear()
        self._nos.clear()

    def add(self, txt, nos):
        """ 添加 """
        self._num += 1
        self._bufs.append(txt)
        self._nos.append(nos)

    def output(self):
        """ 输出 """
        tplt = '{:>10}\t{:>10}\t{:>50}'
        print(tplt.format('序号', '行号', '文本数据'))
        for i in range(self._num):
            print(tplt.format(i + 1, self._nos[i],
self._bufs[i]))
        print('-'*100)


def main():
    path = sys.path[0]
    filePath = path + '/test.txt'
    if os.path.exists(filePath):
        with open(filePath, 'r') as f:
            lines = f.readlines()
        print('读取文本成功...\n' + '-'*100)
        for i, line in enumerate(lines):
            lines[i] = line.replace('\n', '')
        buf = CBuf()
        text = CText(lines, buf)
        text.get_longest()
        text.out.output()


if __name__ == "__main__":
    main()
```

- 运行结果

```
~/Code_Learning(master*) » /home/qu/anaconda3/envs/pytorch/bin/python /home/qu/Code_Learning/Course/面向对象方法学/第三次作业/get_longest.py
读取文本成功...
----------------------------------------------------------------------------------------------------
        序号          行号                                                文本数据
         1             8                                  他们说快写一首情歌雅俗共赏
         2            17                                  是否每一次成熟都徒增了业障
         3            25                                  他们说快写一首情歌雅俗共赏
         4            47                                  太超脱 中枪中奖感觉会一样
----------------------------------------------------------------------------------------------------
```

3. 类和对象的关系

类和对象之间的相互关系的方式包括：**依赖、关联、聚合、组合、泛化、实现** 等

- **依赖**

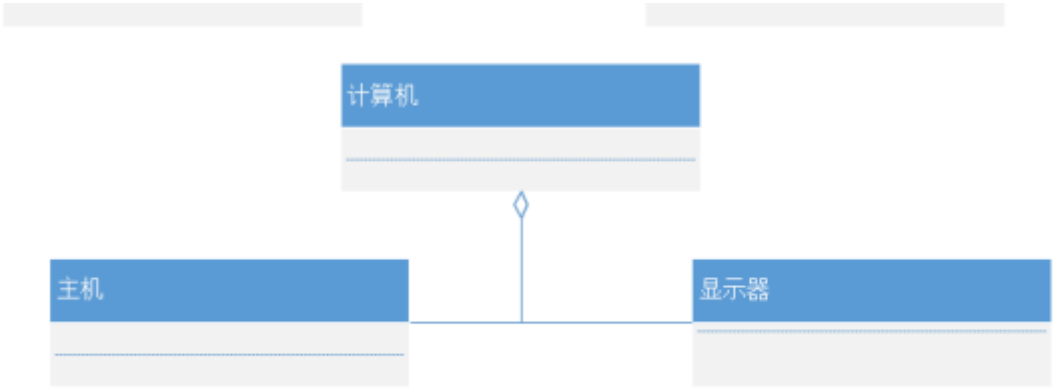
  元素A的变化会影响元素B，但反之不成立，那么B和A的关系是依赖关系，B依赖A；类属关系和实现关系在语义上讲也是依赖关系，但由于其有更特殊的用途，所以被单独描述。

- **关联**

  元素间的结构化关系，是一种弱关系，被关联的元素间通常可以被独立的考虑。给定关联的两个类，可从其中一个类对象访 问到另一个类的相关对象。

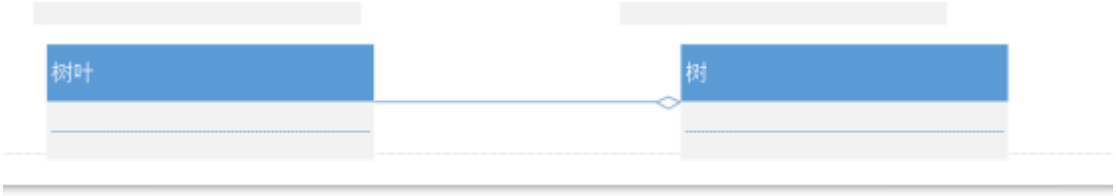
- **聚合**

  关联关系的一种特例，表示部分和整体（整体 has a 部分）的关系。



- **组合**

  组合是聚合关系的变种，表示元素间更强的组合关系。如果是组合关系，如果整体被破坏则个体一定会被破坏，而聚合的个体则可能是被多个整体所共享的，不一定会随着某个整体的破坏而被破坏。



- **泛化**

  通常所说的继承（特殊个体 is kind of 一般个体）关系

- **实现**

  元素A定义一个约定，元素B实现这个约定，则B和A的关系是Realize，B realize A。这个关系最常用于接口。