Object Oriented Programming I

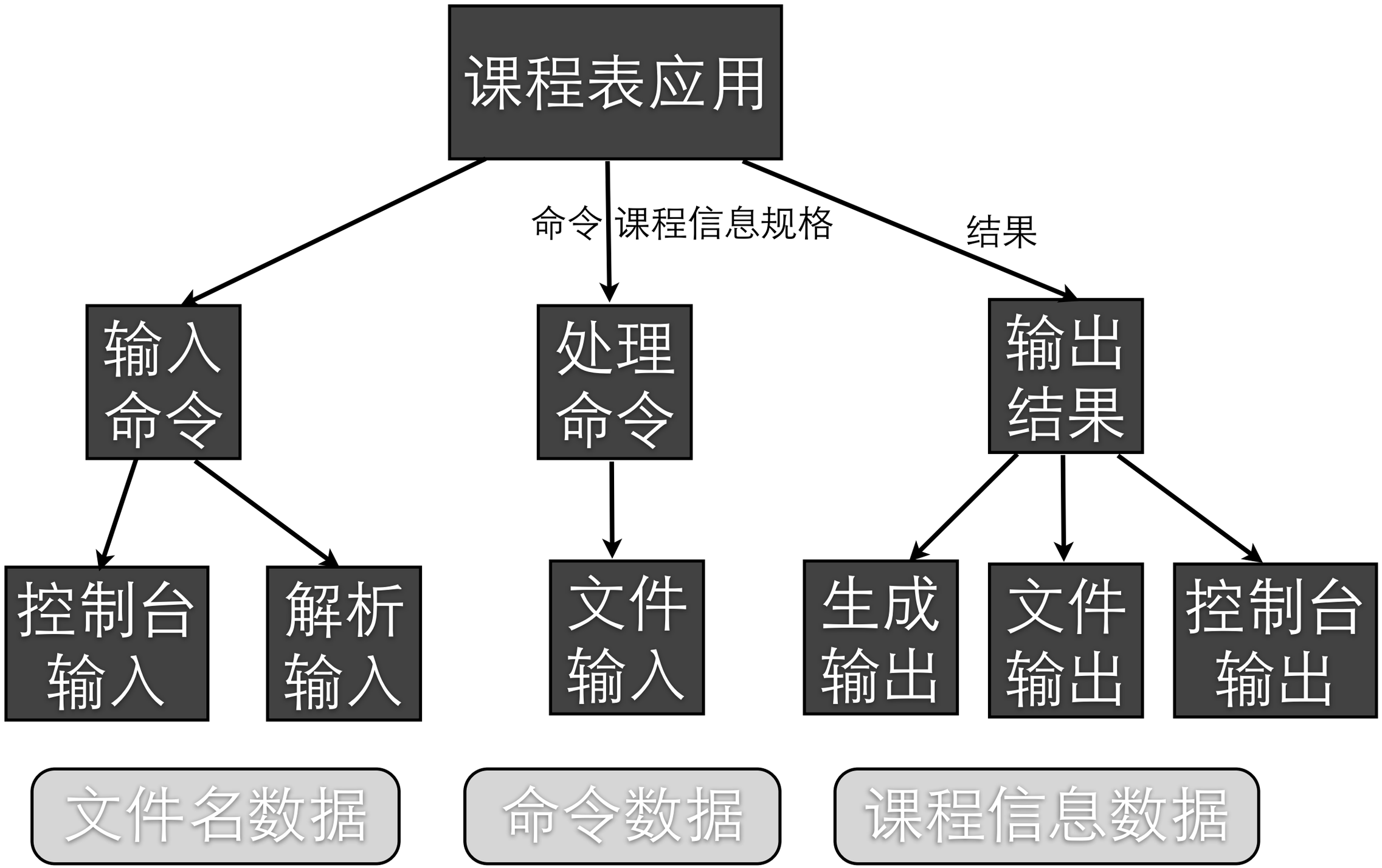
刘 钦

南京大学软件学院

Outline

- Problem of Structured Programming
- Object-Oriented Method
- Object、Class、Responsibility
- Class Diagram and CRC

课程表应用回顾--结构化设计



Structured Programming

- 行为视角
  - 首先根据行为来分解
  - 接着设计数据来配合行为
  - 全局数据

Problems of Structured Programming

- Not easy to read
- Not easy to maintain

Not Easy to Read -- 全局变量

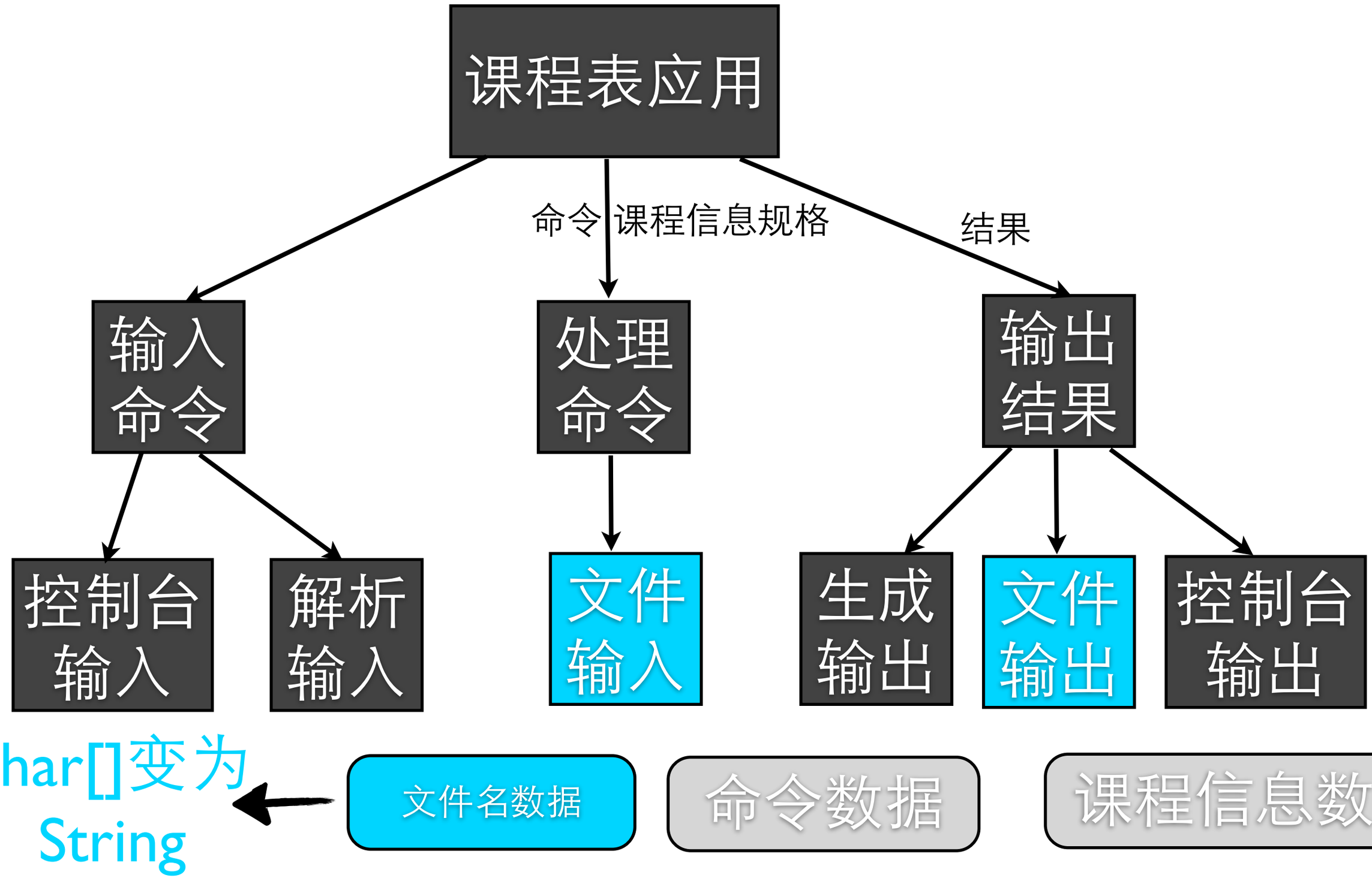
```
public class CourseSchedule {

    static public String fileName = "CurriculumSchedule";

    public static void main(String[] args){
        String input = "",output = "";
        String command;
        String courseInfo;
        int cmd = -1;
```

第11行

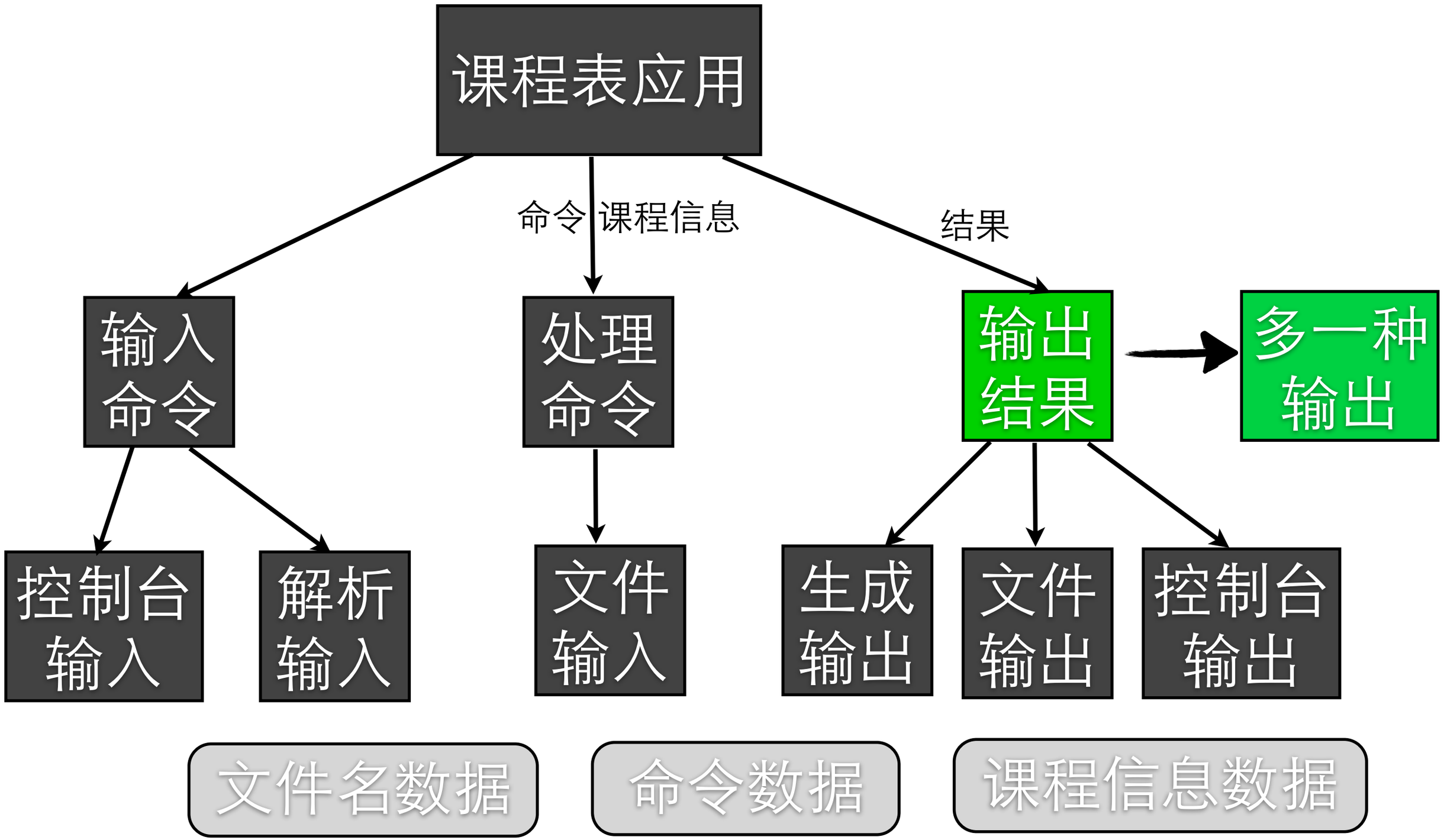
```
//      System.out.println("courseInfo:"+day+time+name+location);
        try{

            BufferedReader br1=new BufferedReader(new FileReader(fileName));
            String line;
            while((line=br1.readLine())!=null){
                String day2;
                String time2;
                String name2;
                String location2;
```

第137行

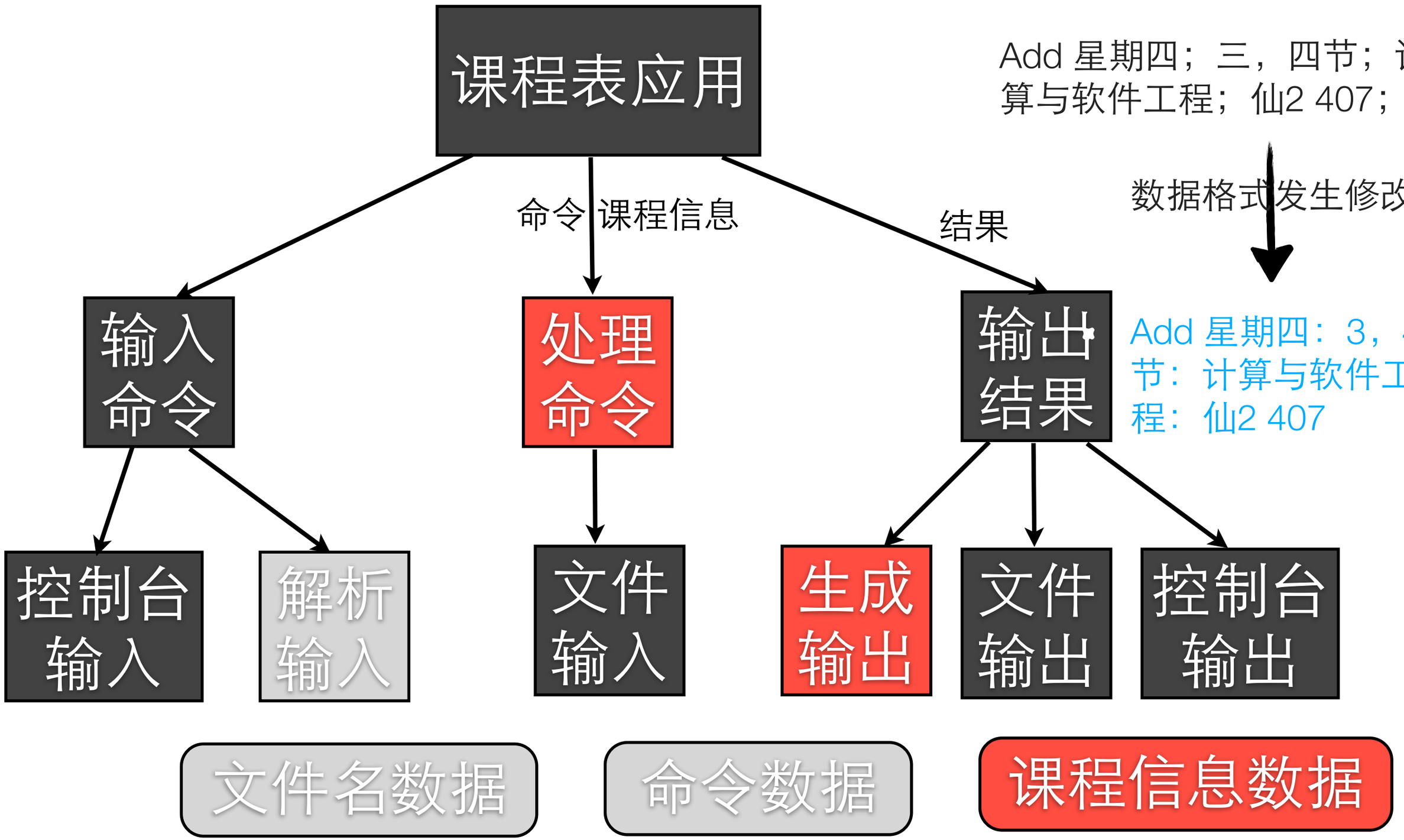
Not Easy to Maintain -- 实现变更



Not Easy to Maintain -- 需求增加



Not Easy to Maintain -- 需求更改



大范围的修改

Nightmare !

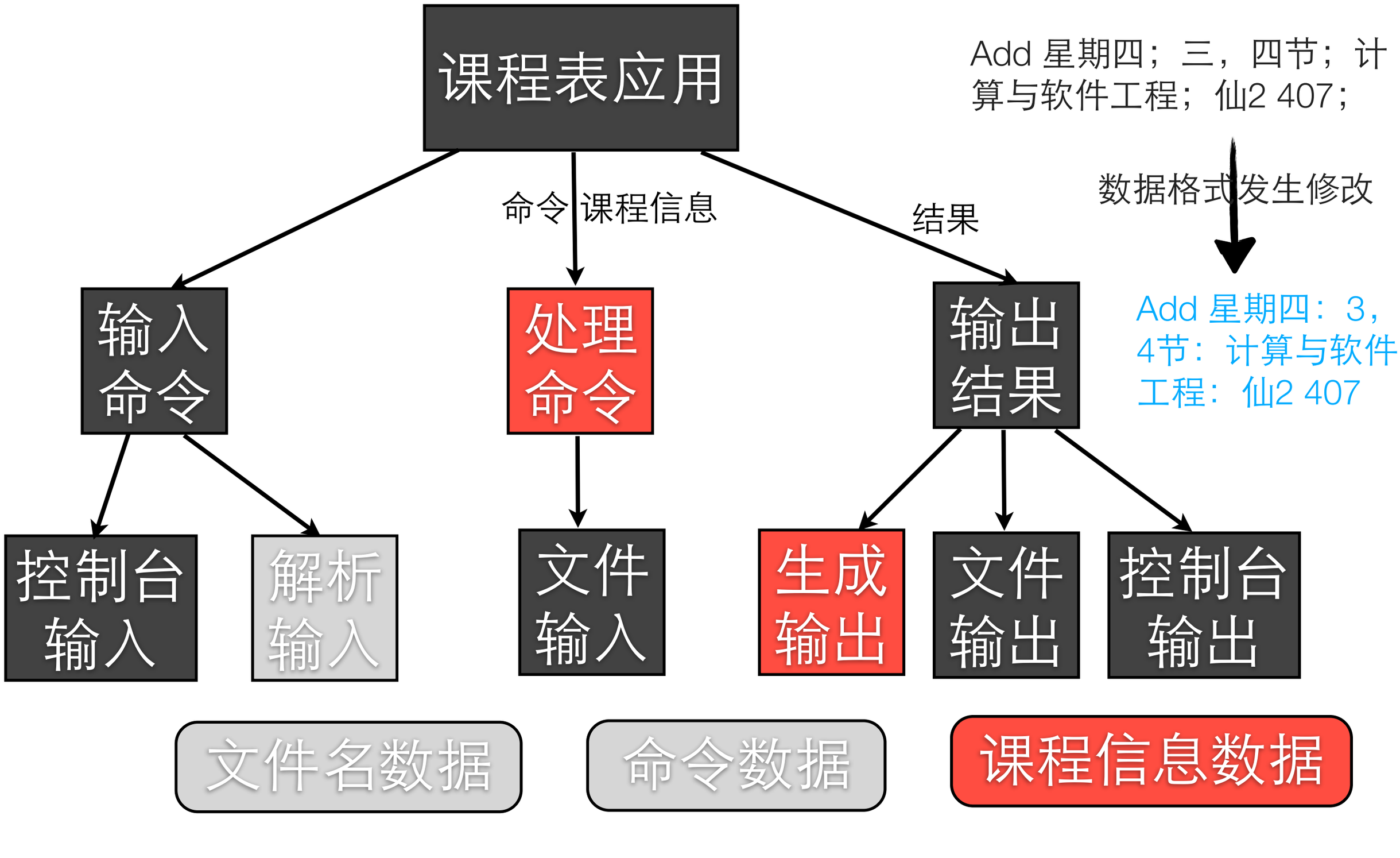
问题

Q: 不修改，怎么应对变更?

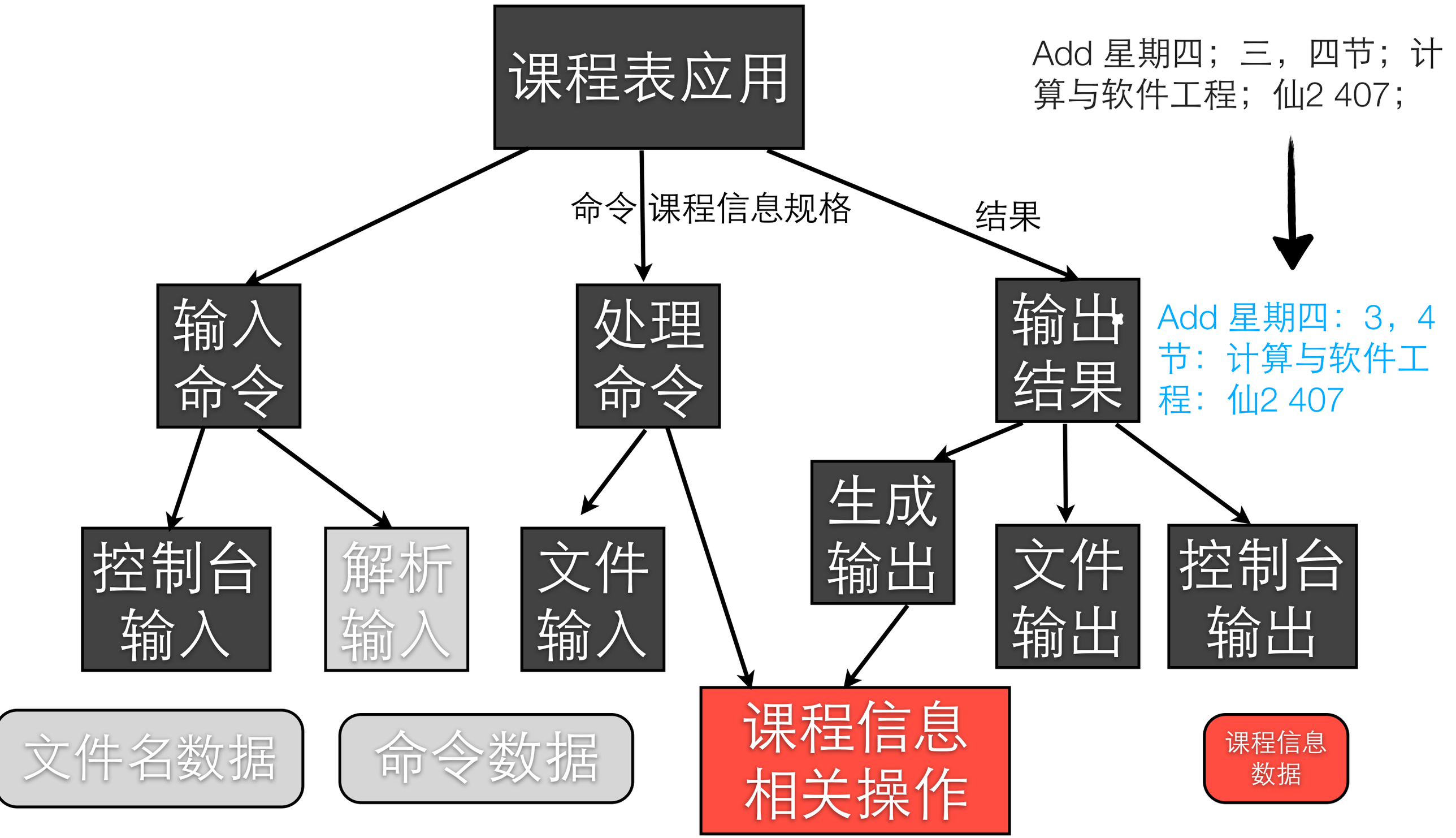
A: 1. 在有限的范围内修改

2. 扩展

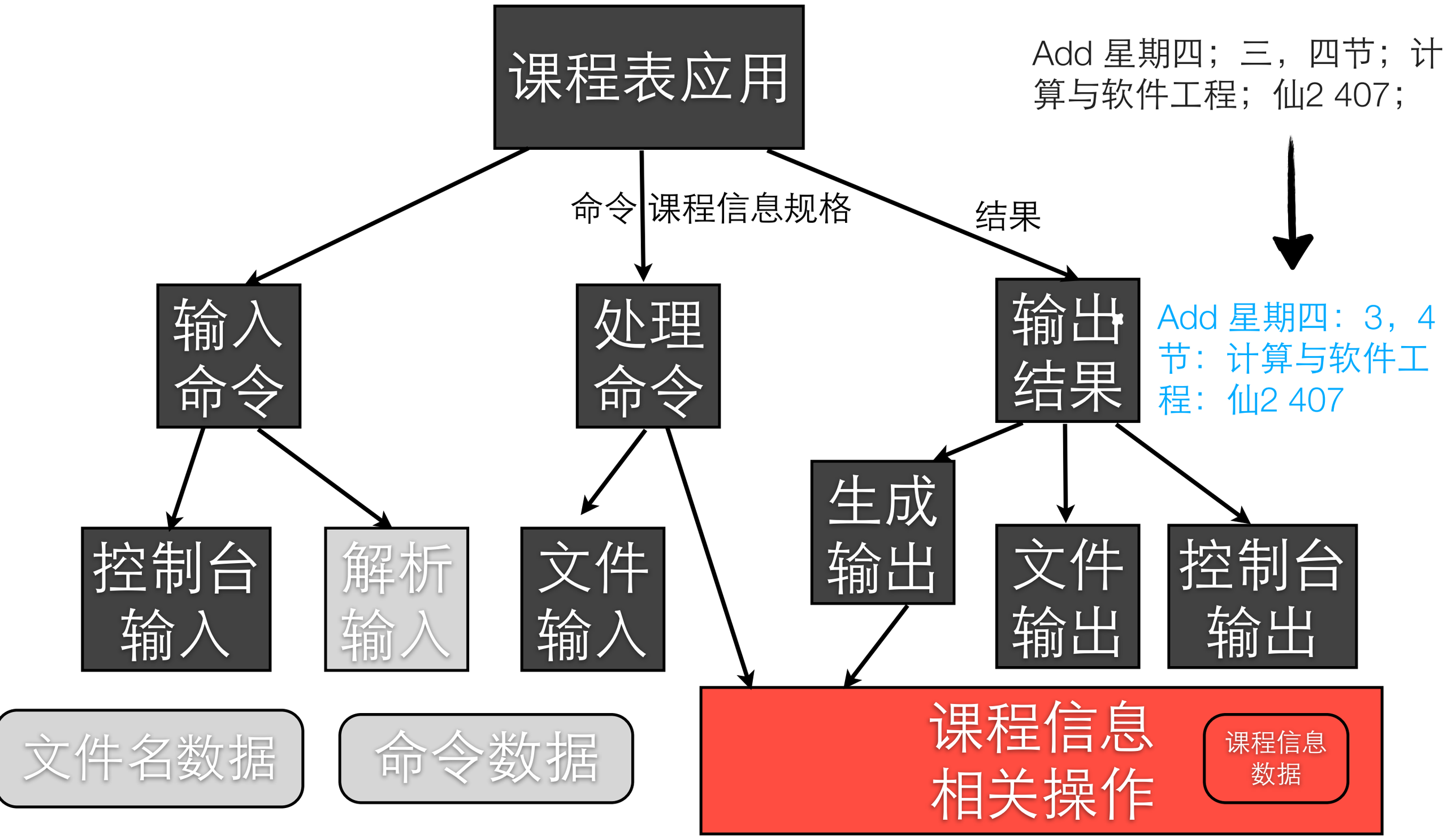
大范围的修改


课程表应用
命令
课程信息
结果
输入命令
处理命令
输出结果
控制台输入
解析输入
文件输入
生成输出
文件输出
控制台输出
Add 星期四；三，四节；计算与软件工程；仙2 407；
数据格式发生修改
Add 星期四：3，4节：计算与软件工程：仙2 407
文件名数据
命令数据
课程信息数据

在有限的范围内修改--相关操作在一起



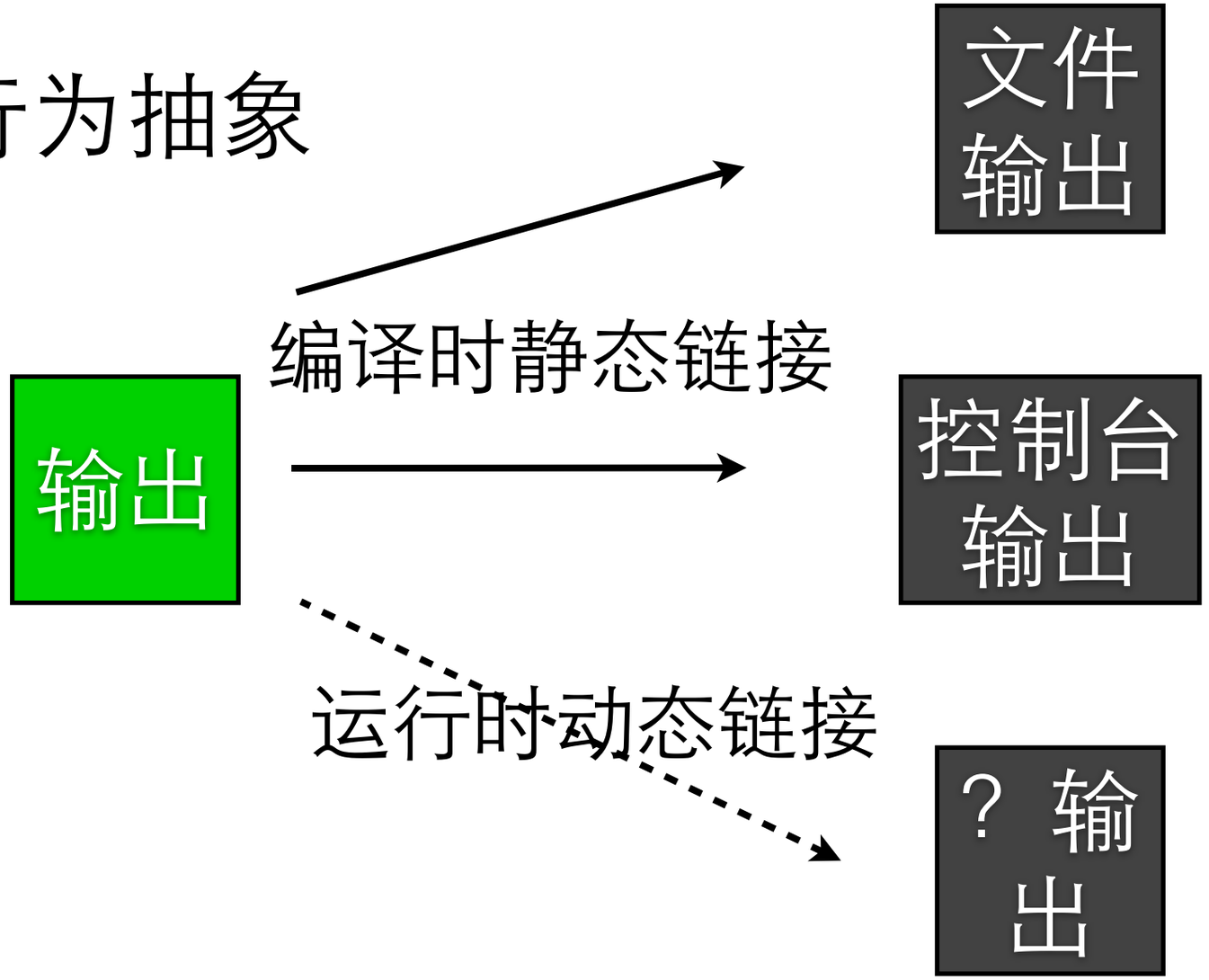
在有限的范围内修改--数据与操作在一起（封装）



扩展--运行时动态链接（多态）

行为实现

行为抽象



职责

职责

- 数据职责和行为职责
- 在一起

再看课程表应用中有哪些职责?

行为职责

- 命令的解析
- 生成输出
- 课程数据的解析
- 控制台输入
- 处理命令
- 文件输入
- 控制台输出
- 文件输出

数据职责

- 课程表数据
- 课程数据
- 命令数据
- 文件数据

职责的分配

命令数据
命令的解析

课程表数据
处理命令
生成输出

课程数据
课程数据的解析

控制台输入
控制台输出

文件数据
文件输入
文件输出

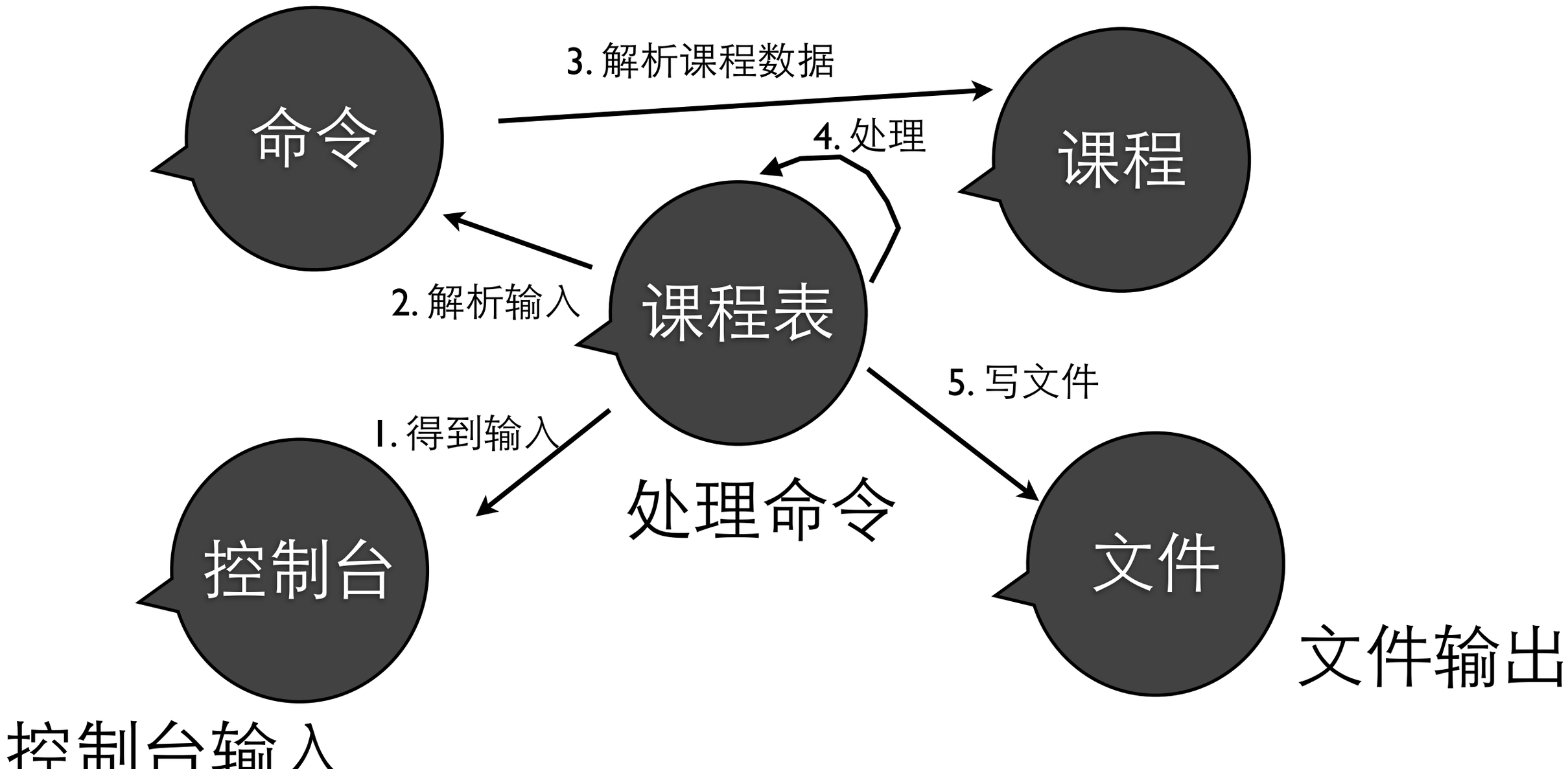
类 - 职责的抽象

光有一个个职责，能不能完成我们的任务！

他们是怎么交互的?

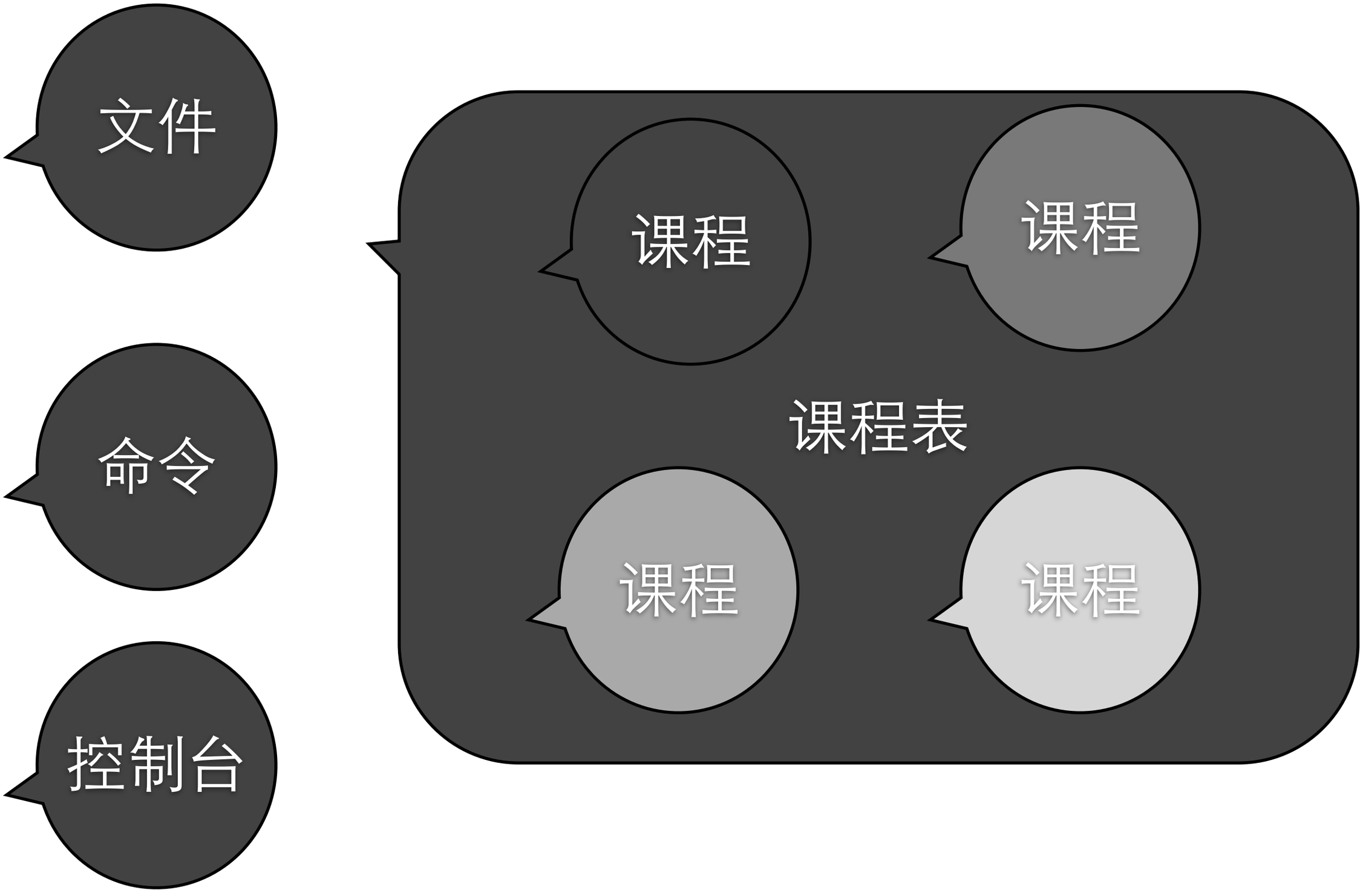
命令的解析

课程数据的解析



控制台输入

课程还有很多个?



对象 - 职责的实现

类和对象的关系

- 抽象与具体

你的世界观改变了！

以前是函数之间的调用，现在是有职责的对象之间的交互

视角的变化

- 行为视角 -- 结构化方法
- 数据视角 -- 数据为中心方法
- 职责视角 -- 面向对象方法

什么是对象

- 对象是面向对象 中的术语，既表示客观世界问题空间(Namespace)中的某个具体的事物，又表示软件系统解空间中的基本元素。
- 在软件系统中，对象具有唯一的标识符，对象包括属性(Properties)和方法(Methods)，属性就是需要记忆的信息，方法就是对象能够提供的服务。

什么是对象

- 每个对象都保存着描述当前特征的信息。
- 对象状态的改变必须通过调用方法实现。
- 对象的状态不能完全描述一个对象。每个对象都有一个唯一的身份(identity)。
- 每个对象的标识永远是不同的，状态常常也存在着差异。

如何获得对象

- 寻找候选对象
  - 找名词 -- 类（对象）与属性
  - 找动词 -- 行为
- 精化对象
  - 去除
    - 冗余
    - 不相干
    - 模糊的概念
  - 转化
    - 没有行为的对象-》某个类的属性

什么是类?

- 类这个术语被用来描述相同事物的集合。它以概要的方式描述了相同事物集合中的所有元素，但却允许类中的每一个实体元素可以在非本质特征上变化。
- 面向对象程序设计语言使用类来描述对象，并且通过类方法来定义它们的行为。

什么是类?

- 类（Class）这个术语是对具有共同具体属性的对象的描述。
- 类是一个描述或蓝图（被表示成一段代码），用于定义组成某类特定对象的所有特性。
- 编程中使用类的思想与现实时间中把东西进行分类的思想相一致，这是一种方便而明确的事物组织方式。

什么是类?

- 一旦定义了一个类，就可以接着得到这个类的对象或实例。
- 实例变量的值由类的每个实例提供。
- 当创建类的实例时， 就建立了这种类型的一个对象，然后系统为类定义的实例变量分配内存。

类与对象

- 类是对某个对象的定义。
- 它包含有关对象动作方式的信息，包括它的名称、方法、属性和事件。
- 当引用类的代码运行时，类的一个新的实例，即对象，就在内存中创建了。虽然只有一个类，但能从这个类在内存中创建多个相同类型的对象。

职责

- 所谓职责，我们可以理解它为功能。
- 每个类应当只有单一职责。
    - 当你发现有两个变化会要求我们修改这个类，那么你就要考虑撤分这个类了。
- 给对象分配责任的策略：
    - 覆盖到所有重要的方面
    - 寻找需要执行的动作以及需要维护和生成的信息

创建类的原因

- 对现实世界中的对象建模
- 对抽象对象建模
- 降低复杂度
- 隔离复杂度
- 隐藏实现细节
- 限制变化所影响的范围
- 创建中心控制点

抽象

- 描述了一个对象的基本特征，可以将这个对象与所有其他类型的对象区分开来，因此提供了清晰定义的概念边界，它与观察者的视角有关

分类

- 分类的重要性
  - 分类的困难
  - 分类的增量和迭代本质

经典方法和现代方法

- 经典分类
  - 所有具有某一个或某一组共同属性的实体构成一个分类。这样的属性对于定义这个分类是必要的也是充分的。（已婚人士，高的人）
  - 属性可以不只表示可以测量的特征，也可以包含观察到的行为(鱼能飞)
- 概念聚集
  - 类的产生首先是形成类的概念描述，然后再根据这些描述对实体进行分类。（爱情歌曲）
- 原型理论
  - 以游戏为例分类可以扩展，新的“游戏”被引入，只要它们以某些恰当的方式与以前的“游戏”表现出相似性

面向对象分析

- 经典方法
  - 来自于问题域的需求
  - 关注问题域中实实在在的事物
- 行为分析
  - 关注动态的行为
  - 强调责任
  - 责任代表的是“对象维护的知识和可以执行的动作，其意义在于表达一个对象的目的以及他在系统中的位置。对象的责任是针对它支持的所有契约而提供的全面服务。
- 领域分析

面向对象分析

- 用例分析
- CRC卡
- 非正式英语描述
- 结构化分析

MediaStudio

Responsibilities	Collaborators
• Manage the script	ScriptController: run
• Manage the animation	
• Display animation	Screen

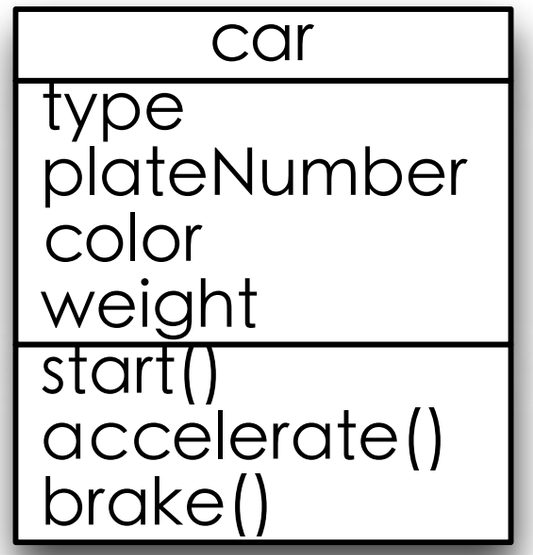
分类

- 关键抽象
  - 一个类或对象，它是问题域词汇表的一部分
    - 给出了问题的边界，突出了系统中的事物
    - 排除了系统之外的事物
    - 确定关键抽象是与 具体领域高度相关的
    - 通过发现和发明来确定关键抽象
- 机制
  - 关于一组对象如何写作的设计决策
  - 代表了行为的模式

类的定义

- class MyClass {
- // field, constructor, and
- // method declarations
- }

类图

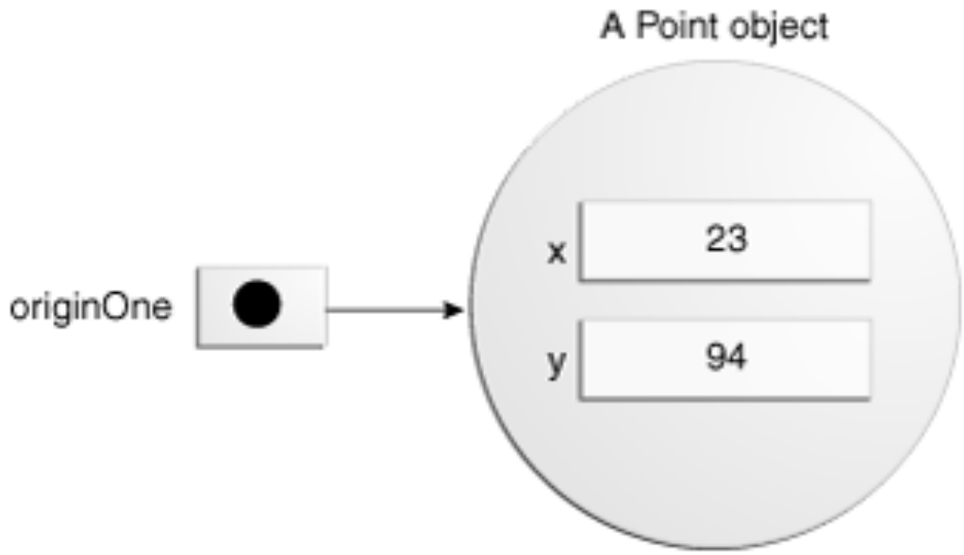


类的定义

```
public class Bicycle {

    private int cadence;
    private int gear;
    private int speed;

    public Bicycle(int startCadence, int startSpeed, int startGear) {  //构造方法
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    public int getCadence() {
        return cadence;
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }

    。。。

    public int getSpeed() {
        return speed;
    }

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }
}
```

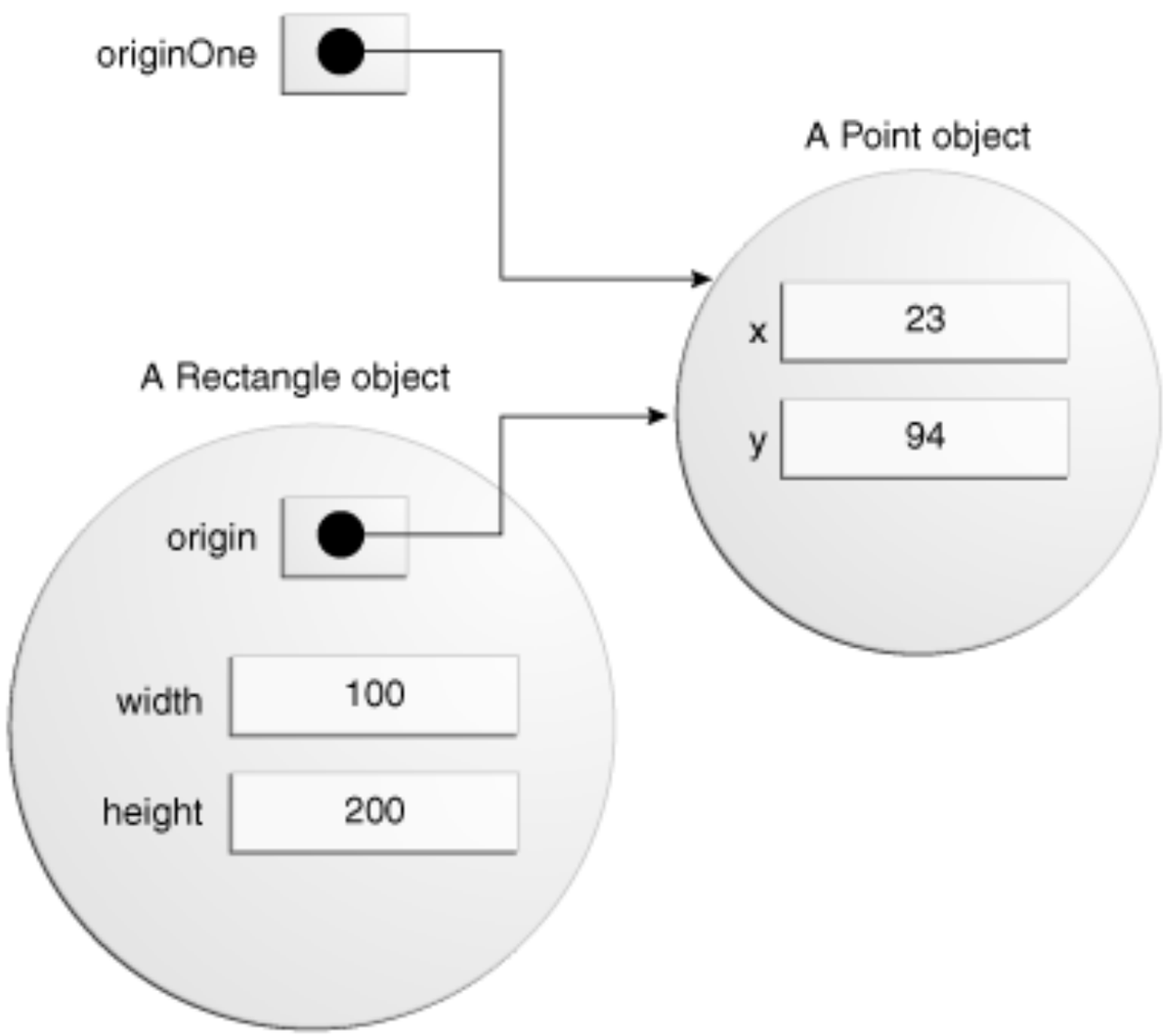
类的定义

- public class Point {
- public int x = 0;
- public int y = 0;
- //constructor
- public Point(int a, int b) {
- x = a;
- y = b;
- }
- }



- Point originOne = new Point(23, 94);

```
public class Rectangle {
    public int width = 0;
    public int height = 0;
    public Point origin;

    // four constructors
    public Rectangle() {
        origin = new Point(0, 0);
    }
    public Rectangle(Point p) {
        origin = p;
    }
    public Rectangle(int w, int h) {
        origin = new Point(0, 0);
        width = w;
        height = h;
    }
    public Rectangle(Point p, int w, int h) {
        origin = p;
        width = w;
        height = h;
    }

    // a method for moving the rectangle
    public void move(int x, int y) {
        origin.x = x;
        origin.y = y;
    }

    // a method for computing the area
    // of the rectangle
    public int getArea() {
        return width * height;
    }
}
```



变量和方法的访问

- 引用变量.变量名
  - 1 . int height = new Rectangle().height;
  - 2. Rectangle rect = new Rectangle();
  - int height = rect.height;
- 引用变量.方法名（）；
  - System.out.println("Area of rectOne: " + rectOne.getArea());
  - ...
  - rectTwo.move(40, 72);

什么是变量？