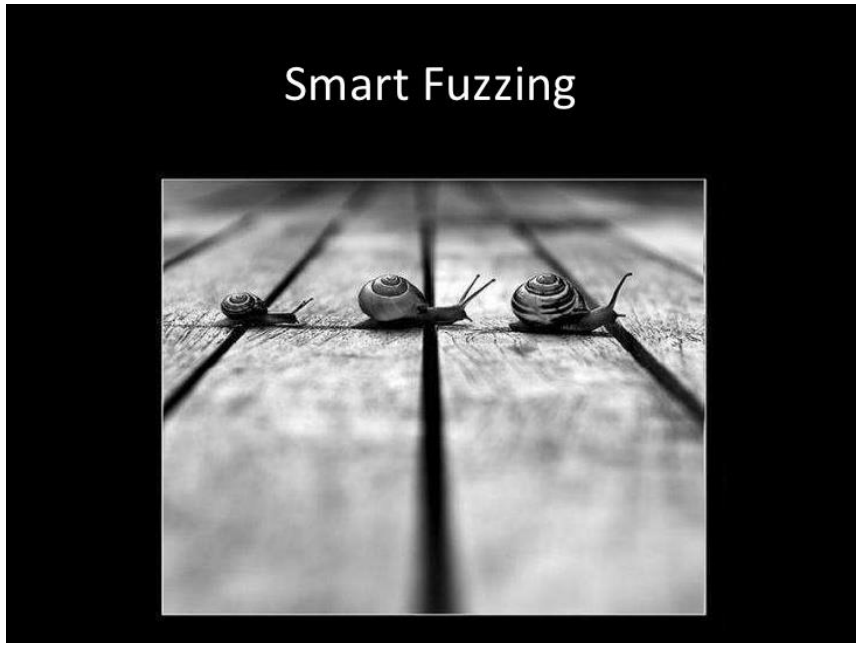
How to conduct effective tests?

- 设计输入来导致程序不能完成其需求所要求提供的处理和输出
  - 按照需求，对输入进行合理划分，形成输入划分层次（树）
  - 覆盖是基本策略，在覆盖基础上逐渐找到重点/弱点，即可能发现bug的输入区域
- 了解程序的结构和算法逻辑，大致确定bug的可能范围
  - 通过分析程序的执行流程和调用路径，与输入建立关系
- 分析程序所使用的数据管理、输入处理结构和类库，了解其可能的局限性和不适用场景
  - 构造相应的输入特征

How to conduct effective tests?

- 道路千万条，安全第一条。。。
  - N-1条呢？
- 如何在正常(normal/regular)输入基础上，自动构造出abnormal输入？



How to conduct effective tests?

- 高质量的程序标志
  - 稳定，不会crash
  - 准确完成要求的功能
  - 能够识别异常输入
  - **代码逻辑清晰和简单**
- 如何设计测试输入？
  - 程序的输入内容是什么(分析作业要求)
  - 程序的输入格式是什么(分析作业要求)
  - 程序对输入做什么处理(阅读代码)
  - 程序的输出结果是什么(观察程序运行的反馈)

关于代码风格

- 代码风格检查强调的不只是代码好看
- 设计规范
  - 类规模
  - 方法规模
  - 输入数据复杂度
  - 属性可见性
  - 布尔表达式复杂度
  - 禁止对输入参数赋值
  - 循环及条件嵌套重数限制

《面向对象设计与构造》课程 Lec2-对象与对象化编程(下)

2019

OO课程组

北京航空航天大学

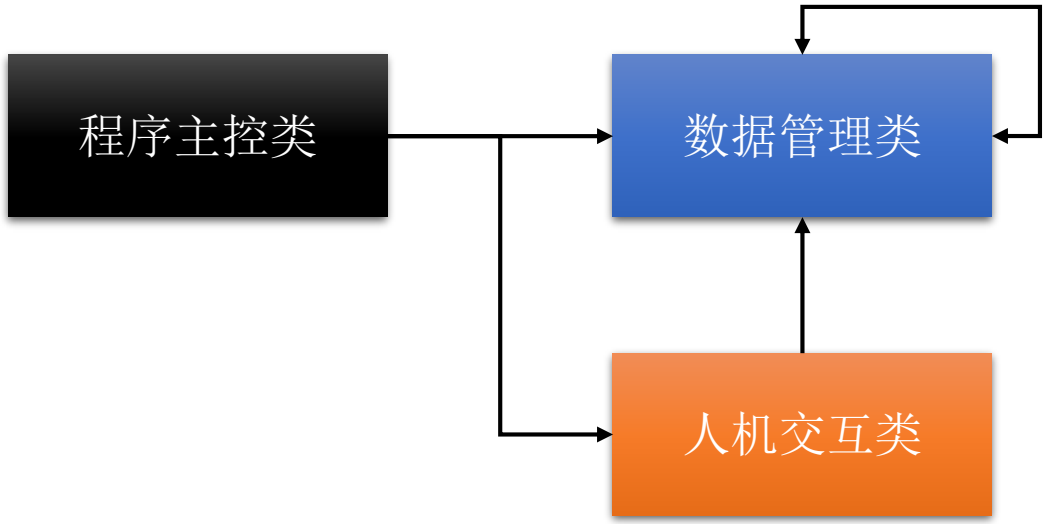
内容提要

- 认识对象的特性
- 对象的可变性
- 类的属性与方法
- 两类Java程序
- 作业

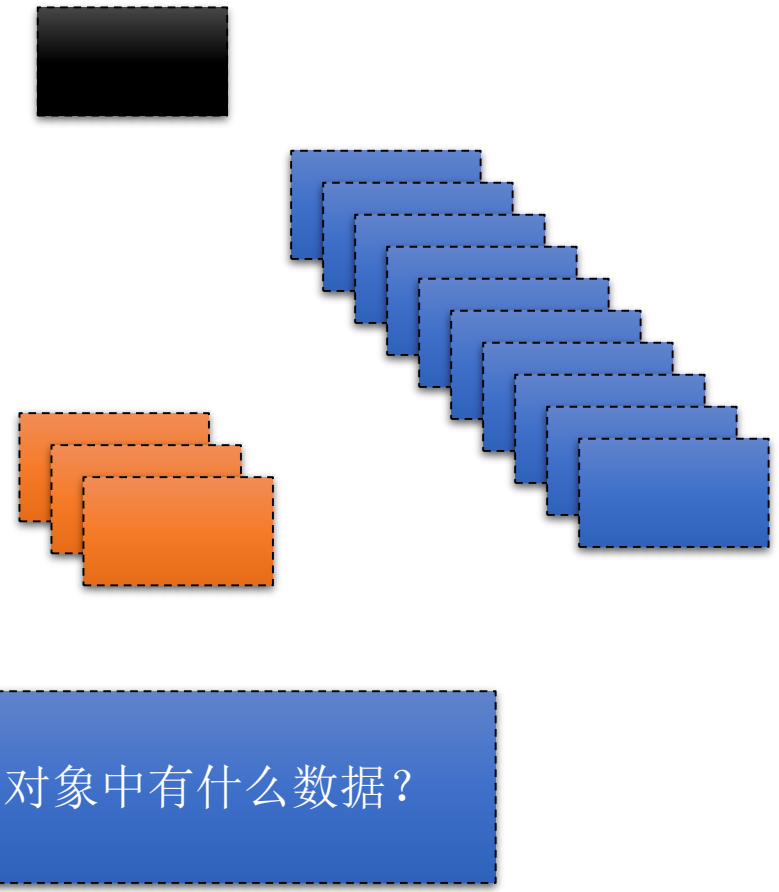
再次明确几个核心概念

- 对象
  - 存储在内存中的一个数据结构。任意时刻都拥有确定的取值。该数据结构由相应的类定义。
- 对象属性
  - 对象组成成分，有类型和具体取值。可以是简单类型，或者复杂类型（类）。
  - 对象属性具有可见性设定，用来确定能够访问该属性的对象集合。
- 对象方法
  - 对象的行为操作，由相应类定义。
- 对象引用(object reference)
  - 由相应类定义的变量指向内存中某个类型相匹配的实际对象的结果。
  - 要与被引用的对象(referred object)加以区分
- 对象访问
  - 通过对象引用来使用一个对象的行为。根据可见性设定，可以访问对象属性或者对象方法。

代码(类)空间



运行时(对象)空间



认识对象的特性

- 对象是运行时的存在，具备多种特性
  - 静态特性：由相应类所定义的特性，运行时保持不变
  - 动态特性：对象运行时获得、且可能会动态变化的特性
- 静态特性
  - “任何三角形对象的三个顶点都不会在一条直线上”
  - “任何储蓄账户对象的余额必须不会小于0”
- 动态特性
  - “一部车的燃油消耗由行驶里程和行驶速度决定”
  - “一个程序运行时使用的内存随输入发生变化”
- 对象特性本质上是关于对象属性取值或关系的特性
  - 对象可变性是一个关于对象属性是否会发生变化的特性

对象的可变性

可变对象中可能包括不可变数据

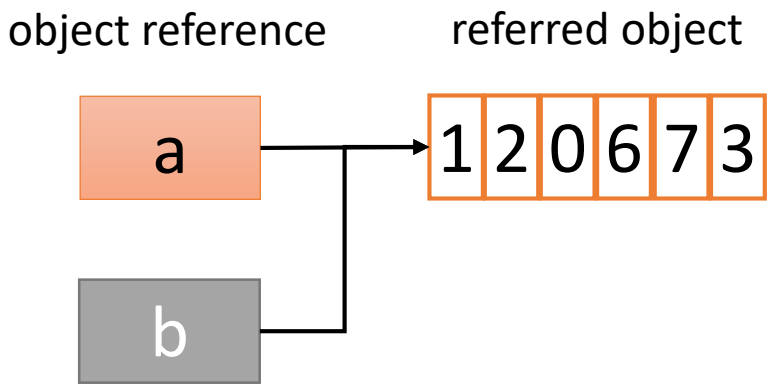
- 可变对象(mutable object)
  - 状态可发生外部能够观察到的变化
- 不可变对象(immutable object)
  - 对象的属性不可以被改变
  - 或者对象的属性可以被改变，但是外部观察不到相应状态的变化
  - 典型代表：常量字符串对象
- 使用不可变对象能够降低逻辑复杂度，易于发现问题
  - 可能会导致内存消耗多

```
public class Poly{
    private int[] terms;
    private int deg;
    public Poly(int deg) {…}
    public Poly(int c, int n){…}

    public int degree(){return deg;}
    public int coeff(int d){…}

    public Poly add(Poly q){…}
    public Poly sub(Poly q){…}
}
```

mutable or immutable object?

对象可变性

- 在不引起歧义情况下，有时也可称“对象引用”为“**对象”
  - ArrayList a;
  - if(a.get[i] ==0)…
  - a.set(i, 1);
- 多个对象引用可指向一个相同对象
  - ArrayList b = a;
- 多个指向同一对象的引用实际产生了对象共享
  - 如果被引用对象不可变，共享不会产生风险
  - 如果被引用对象可变，共享有可能产生不可预期的效果
    - b.set(3, 9)



对象可变性

- 共享访问可变对象可能产生的风险
  - 一个对象所管理的数据被外部某对象不受控制的访问
    - 如账户管理对象如果把账户相关数据暴露给外部，可能会破坏账户的关键信息
  - 一个临时对象被共享，导致其生命期结束时间不确定
    - Java程序中，一个方法构造的临时对象在方法执行结束后其生命期就结束。一旦与外部对象产生共享，其生命期结束时间就不确定，相应的内存被回收时间也不确定
  - 如果被多个线程共享访问，运行时可能会出现莫名错误
    - 如多个储户线程同时访问（存、取）一个账户对象，会导致账户余额发生莫名其妙的变化。

对象共享与对象复制

- 对象引用复制即产生共享
- 如果想要复制对象，而不是共享
  - 使用对象的clone方法产生新的对象
  - 或者，new一个新对象，拷贝对象的属性值（要求对象的属性值外部可访问）

```
public class Poly{
    private ArrayList terms;
    private int deg;
    …
}
```

```
public class Poly{
    private ArrayList terms;
    private int deg;
    public Poly clone(){
        Poly p = new Poly();
        p.terms = this.terms;
        p.deg = this.deg;
        return p;    }
}
```

Java根对象

- Java语言预定义了一个根类Object，除了原子数据类型(primitive type)，任何类(包括用户定义的类)都默认是Object的子类
- Object提供了三个重要方法
  - boolean equals(Object o)
  - Object clone()
  - String toString()

为什么要设计这个Object？

(1)需要一个统一的类型体系(概括能力)
(2)需要在运行时处理所有对象的能力

对象等同判断

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- 很多时候程序需要判断两个对象的关系
  - *类型是否相似*
  - *类型是否相同*
  - 状态是否相同
  - 对象是否相同

对象相同→对象状态相同→对象类型相同→对象类型相似

```
String s1 = "hello";
String s2 = new String("hello");
System.out.println(s1==s2);
System.out.println(s1.equals(s2));
```

- 如果一个对象，两个对象是相同对象（即使用Object的equals方法来
- 如果一个对象同，就应该认为是相同对象，这时需新实现equals方法

对Object方法的实现

- 可变对象
  - 需要实现*clone*
  - 需要实现toString
- 不可变对象
  - 需要实现equals
  - 需要实现toString

```
public class Poly{
    private ArrayList terms;
    private int deg;

    //method definitions…
}
```

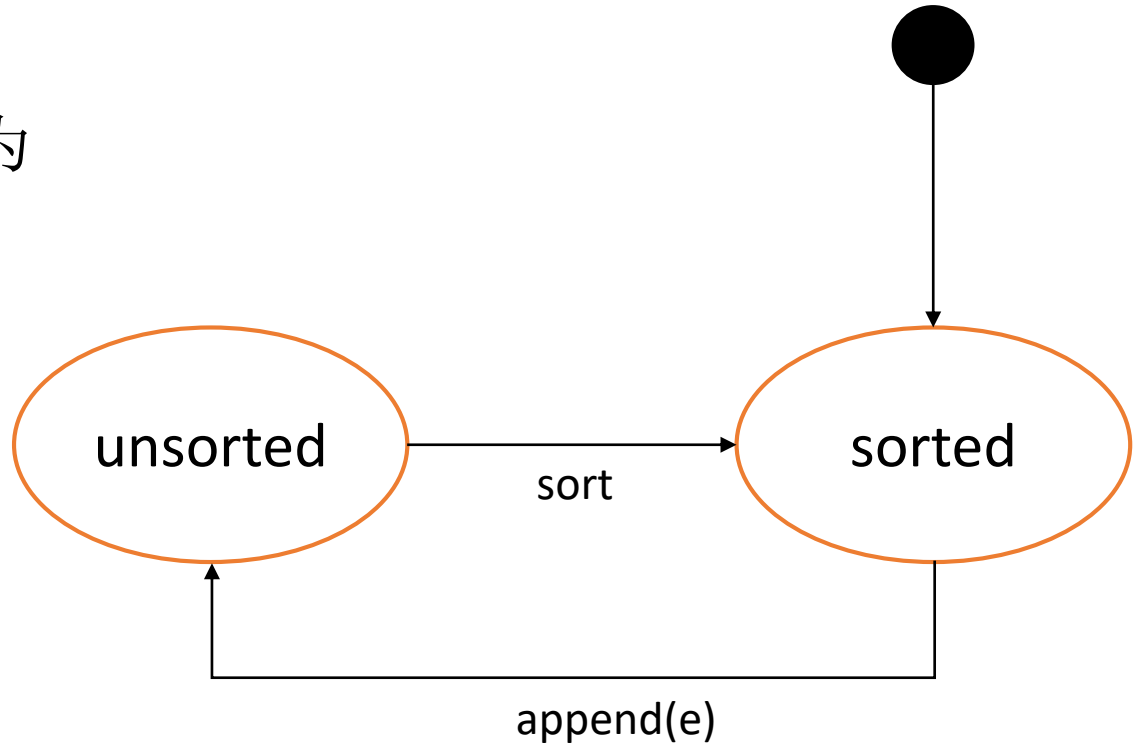
toString：输出多项式的内容

Poly对象可以是可变对象
也可以是不可变对象
请用10分钟分别给出相应的片段代码

对象属性与方法

- 属性与方法构成了类的实现细节
  - 属性定义对象状态
  - 方法定义对象作用于其状态上的行为

```
public class ScalableArray{
    private Vector els;
    public void append(int e){els.add(e);}
    public void sort(){…}
}
```

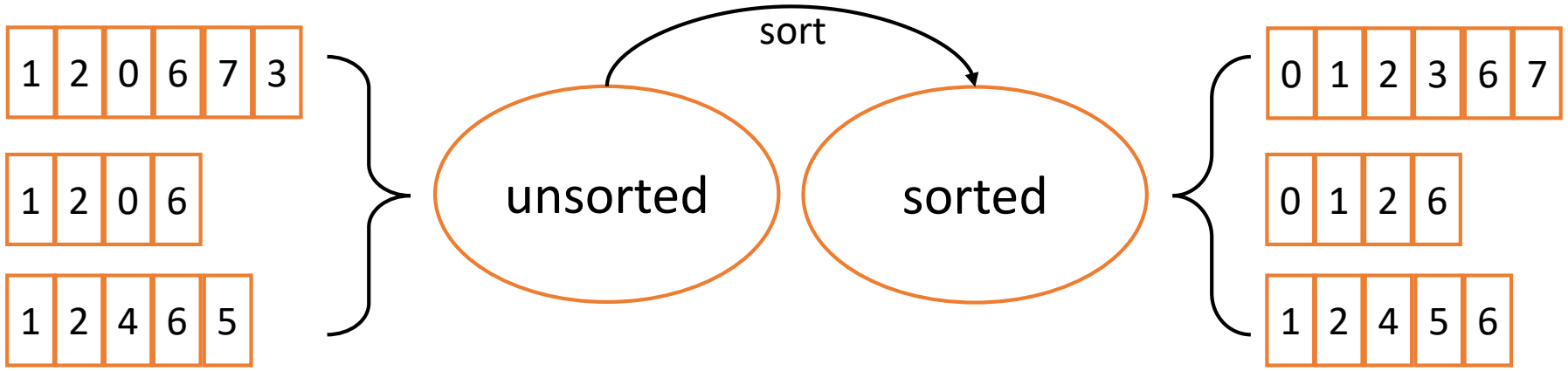


(1)该类隐藏了哪些状态？
(2)append是否一定会改变状态？
(3)如果append不改变状态，这个状态图如何调整？

对象属性与方法

- 状态
  - 内部状态 (memory → state)
    - 向量els中的所有元素
    - els中元素的顺序关系
  - 外部状态 (internal state → external state)
    - unsorted：存在i<j<k, (els[i] - els[j])*(els[j] - els[k]) <0
    - sorted: 任意i<j<k, (els[i] - els[j])*(els[j] - els[k]) >= 0

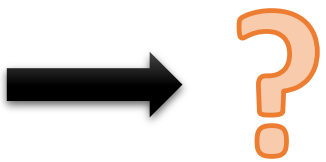
```
public class ScalableArray{
    private Vector els;
    public void append(int e){els.add(e);}
    public void sort(){…}
}
```



对象属性与方法

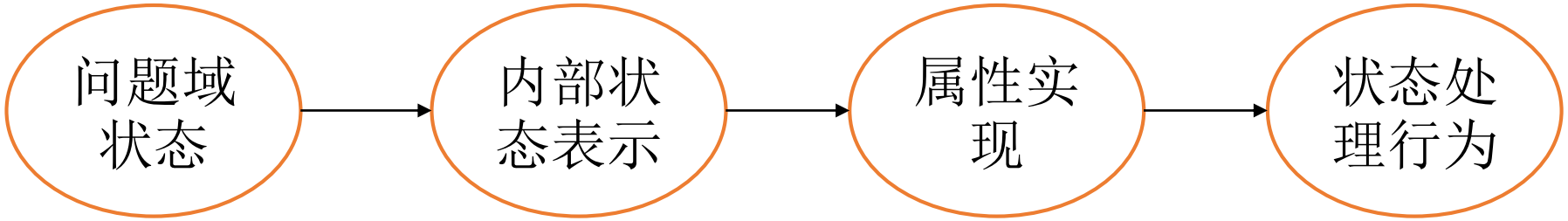
- 如果一个对象确保其外部状态始终保持不变，则可视为immutable object
- 例如，如果要求ScalableArray始终处于Sorted状态
  - 该类的行为如何调整？

```
public class ScalableArray{
    private Vector els;
    public void append(int e){els.add(e);}
    public void sort(){…}
}
```



对象属性与方法

- 定义对象的状态
  - 外部状态影响对象交互行为
  - 内部状态影响对象的计算处理行为
- 依据需求中描述的问题域特征来定义状态
  - 电梯
  - 图书馆的书
  - 手机联系人
- 对象状态可用于定义属性和相应的方法



对象属性与方法

- 可见性(visibility)
  - 用于访问控制
  - OO基本准则：隐藏尽可能多的细节
  - private: 仅限相同类的对象内部访问（**可跨同类对象访问**）
  - public: 对外部完全公开（**可跨任意对象访问**）
  - protected: 仅对当前对象和子类对象公开
- 属性与方法的修改影响(change impact)
  - 应尽量保持private，对其修改外部类不可见
  - protected：对其修改后可能需要修改子类实现
  - public：对其修改需要对任何使用相应对象的实现进行修改

对象属性与方法

- 一个对象该有哪些属性：依从性问题
  - 原则1：逻辑依从
    - 类的属性应该是能够在问题域层次可见的属性（依从问题域层次状态的表示需要）
      - 电梯类、集合类、学生类、三角形类
  - 原则2：计算效率依从
    - 用于提高某些方法计算效率的属性，常常是推导属性(derived)
      - 数组长度、数组最大值、最小值
      - 三角形类别、集合是否为空
      - 学生已修学分总数

对象属性与方法

- 针对给定类的每个属性
  - 如果去掉，该类所要存储和管理的数据是否完整？
  - 如果去掉，该类的行为受到什么影响？
- 关键
  - 确定类的逻辑/状态边界 → 数据边界
  - 实践中，很容易把应该属于其他类的属性放到当前类中
    - 如图书馆系统的读者类
  - 如果多个类在逻辑上涉及相同的数据怎么办？
    - 网络论坛系统，帖子(Post)类与帖子阅读或回复通知消息类(Message)
    - 网络叫车系统(“嘀嘀”)，出租车调度类与出租车类

对象属性与方法

- Case study
  - 学生成绩管理系统，功能包括选课、填报成绩、查询成绩、统计学分。
  - 请大家使用10分钟时间来整理Student, Course这两个类的属性

```
public class Student {
    private String stuID;
    private String stuName;
    private StuKind kind;
    private float totalcredits;
    private Course list[];
}
```

如何了解哪些同学选了某门课？

如何描述一个同学重修某门课？

是否需要了解一个同学什么时间修了某门课？

成绩作为Course的属性是否合适？

```
public class Course{
    private String courseID;
    private String courseName;
    private CourseKind kind;
    private float credit; //学分
    private float mark; //成绩
}
```

对象属性与方法

- Case study
  - 学生成绩管理系统，功能包括选课、填报成绩、查询成绩、统计学分。
  - 请大家使用10分钟时间来整理Student, Course这两个类的属性

```
public class Student {
    private String stuID;
    private String stuName;
    private StuKind kind;
    private float totalcredits;
    private CourseSelection list[];
}
```

```
public class CourseSelection{
    private Student student;
    private Course course;
    private Semester sem;
    private boolean reselection;
    private float mark; //成绩
    private float credit; //获得的学分
}
```

```
public class Course{
    private String courseID;
    private String courseName;
    private CourseKind kind;
    private float credit; //学分
    private float mark; //成绩
}
```

学生如何查看有哪些课可选？

CourseSelection list定义为普通数组还是容器对象？

CourseSelection是否需要保存学分？

对象属性与方法

- 集合类属性
  - 某些属性需要存储多于一个相关的元素或对象（如CourseSelection list[]）
  - 规模已知（如100*100的图片）
    - 使用静态定长数组
  - 规模动态时确定，且保持不变
    - 使用动态数组，运行时申请内存
  - 规模动态时确定，且动态变化
    - 使用Vector、ArrayList等具有伸缩性的容器

对象属性与方法

- 一般而言，方法包括如下三种
  - 构造方法
    - 设置属性的初始值，即设置对象初始状态
  - 状态查询方法
    - 返回内部状态（即**属性值**）
    - 返回外部状态（执行内部状态到外部状态的转化）
  - 状态改变方法
    - 针对功能要求改变内部状态

对象属性与方法

- 构造方法
  - 缺省构造方法，不需要输入任何参数
    - 把原子类型变量设置为默认初值，如0，false等
    - 设置对象变量（如果无法构造，则设为null）
    - 设置容器类变量（一般初始化为empty状态）
  - 针对多种情况的构造方法，需要相关参数

构造方法什么时候被调用？

```
public Poly (int c, int n) throws NegativeExponentException {
   // EFFECTS: If n < 0 throws NegativeExponentException else
   //    initializes this to be the Poly cx^n.
   if (n < 0)
      throw new NegativeExponentException("Poly(int,int) constructor");
   if (c == 0) { trms = new int[1]; deg = 0; return; }
   trms = new int[n+1];
   for (int i = 0; i < n; i++) trms[i] = 0;
   trms[n] = c;
   deg = n; }
```

对象属性与方法

- 状态查询方法
  - 盲目为每个属性添加一个get和set方法是没有意义的
  - 这类方法容易导致对象共享
    - 无意识的共享
  - 有些属性的取值不能允许外部访问
    - 账户对象的密码属性
  - 有些属性不能允许外部设置
    - 如电梯运行方向属性

对象属性与方法

- 状态查询方法+状态改变方法
  - 启发式规则1：信息专家
    - 拥有相应属性的类就是相关信息的专家
    - 其他类需要向这个“专家”咨询什么？
    - “专家”需要对外界请求做哪些状态更新？
    - 例：学生成绩管理系统中的Student类

其他类需要它
(1)选一门课
(2)退选一门课
(3)登记一门课的成绩

其他类需要了解
(1)一个学生是否选了某门课
(2)一个学生的类别（本科生、研究生、…）
(3)学生的姓名
(4)学生总的学分
(5)学生选的某门课的成绩

```
public class Student {
    public boolean courseSelected(Course c){…}
    public StuKind getStuKind(){…}
    public String getName(){…}
    public float getTotalCredit(){…}
    public float getCourseMark(Course c){…}
    public boolean selectCourse(Course c){…}
    public boolean unselectCourse(Course c){…}
    public boolean recordMark(Course c, float m){…}
}
```

对象属性与方法

- 状态查询方法+状态改变方法
  - 启发式规则2：控制专家
  - 一个类被赋予了拥有响应**系统事件**的能力----控制专家
  - 系统事件有哪些？处理这些事件需要哪些信息？
  - 例：电梯系统
    - 电梯类、电梯请求队列类、请求调度类
    - 系统事件：楼层请求（上/下）、电梯运载请求（到达某一楼层）
    - 楼层请求：楼层号、请求方向、请求时间
    - 电梯运载请求：楼层号、请求时间
    - 我们选择让调度类来响应这些请求，响应结果是构造请求对象，放入请求队列；然后按照既定的调度策略来调度电梯响应这些请求。

对象属性与方法

- 当为类设计一个方法时，要明确
  - 这个方法运行后达到的效果是什么
  - 这个方法运行时要求满足的条件是什么
  - 这个方法运行时需要使用哪些数据

```
public boolean recordMark(Course c, float m)
```

```
public class Student {
    public boolean courseSelected(Course c){…}
    public StuKind getStuKind(){…}
    public String getName(){…}
    public float getTotalCredit(){…}
    public float getCourseMark(Course c){…}
    public boolean selectCourse(Course c){…}
    public boolean unselectCourse(Course c){…}
    public boolean recordMark(Course c, float m){…}
}
```

两类Java程序

- 命令行执行程序
  - 需要提供一个显示的入口点public static void main(String args[])
  - 执行过程中只能通过命令行与用户进行交互
  - 我们将首先关注这一类程序
- 基于可视化UI的交互式程序
  - 利用Java提供的UI框架
  - 在适当的位置构造和展示业务对象，并通过HCI事件进行业务处理
  - 需要多线程处理，保持UI能够持续响应用户的请求
  - 这一类程序本质上并无特殊之处，只是需要更多Java类库支持

作业分析

- 本周作业在上次基础上，在幂函数基础上增加简单正余弦函数，继续进行求导操作
  - 增加了新的“项”：正余弦函数
  - 增加了新的求导规则：正余弦函数求导、同类因子的乘积表达式求导
  - 大的结构保持不变（是否可以？）
- 看看你的代码是否需要重构？
  - 如果清楚的区分了项和求导规则，以及二者的适配关系，新增加的需求就很容易处理
  - 如果没有，建议立刻重构，否则下次作业恐怕就很难完成了

代码重构

- 难以通过风格检查中的设计规范时(bad smell)，好的重构时机
- 重构经常解决的问题
  - 长的方法流程：切成片，形成多个方法
  - 重类：分解形成多个类，特别是按照方法与数据的交互关系进行分类
  - 过度中心化的类关系：某些类需要协调所有事情，职责分派出去
- 重构后的关键事项
  - 与重构前的代码做对比测试，要上量，想办法自动化
  - 分析重构效果，结构图对比