面向对象程序设计

主讲：陈笑沙

目录

5.1 抽象

代码的抽象能力是编程中至关重要的核心能力，它决定了代码的可维护性、扩展性和复用性。

5.1 抽象

每一种强有力的语言……提供了三种机制：

- **基本表达形式**：用于表示语言所关心的最简单的个体。
- **组合的方法**：通过它们可以从较简单的东西出发构造出复合的元素。
- **抽象的方法**：通过它们可以为复合对象命名，并将它们当作单元去操作。

——《计算机程序的构造和解释》

5.1 抽象

哲学本质：抽象是对现实的逻辑建模

- 抽象是计算机科学的第一性原理，通过选择性忽略细节，构建事物的本质模型
- 类似于地图绘制：保留主干道路，省略树木和路灯等细节
- 示例：用坐标系表示城市交通，用节点和边表示社交网络关系

5.1 抽象

技术实现的三重境界

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装
  - 例如：函数的抽象

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装
  - 例如：函数的抽象
- 中级抽象：结构建模

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装
  - 例如：函数的抽象
- 中级抽象：结构建模
  - 例如：类的抽象

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装
  - 例如：函数的抽象
- 中级抽象：结构建模
  - 例如：类的抽象
- 高级抽象：模式与范式

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装
  - 例如：函数的抽象
- 中级抽象：结构建模
  - 例如：类的抽象
- 高级抽象：模式与范式
  - 设计模式

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装
  - 例如：函数的抽象
- 中级抽象：结构建模
  - 例如：类的抽象
- 高级抽象：模式与范式
  - 设计模式
  - MVC 分层抽象

5.1 抽象

技术实现的三重境界

- 初级抽象：代码封装
  - 例如：函数的抽象
- 中级抽象：结构建模
  - 例如：类的抽象
- 高级抽象：模式与范式
  - 设计模式
  - MVC 分层抽象
  - 函数式编程中的 Monad 抽象

5.1 抽象

抽象能力的演进路径

5.1 抽象

抽象能力的演进路径

- 初学者：识别重复代码

5.1 抽象

抽象能力的演进路径

- 初学者：识别重复代码
- 进阶者：设计领域模型

5.1 抽象

抽象能力的演进路径

- 初学者：识别重复代码
- 进阶者：设计领域模型
- 专家：创建DSL（特定领域语言）

5.2 分类

5.2 分类

- 面向对象程序设计自定义数据类型，该数据类型自成一体，隐藏了数据组成及操作，体现了程序设计中的抽象性

5.2 分类

- 面向对象程序设计自定义数据类型，该数据类型自成一体，隐藏了数据组成及操作，体现了程序设计中的抽象性
- 自定义数据类型，非孤立(后面会看到继承)，而是系统化分层结构

5.2 分类

- 面向对象程序设计自定义数据类型，该数据类型自成一体，隐藏了数据组成及操作，体现了程序设计中的抽象性
- 自定义数据类型，非孤立(后面会看到继承)，而是系统化分层结构
- 分类是设计和划分类对象的边界的一种手段，它分别作为各个类别的实体，区分其不同类、不同数据属性、不同范围以及不同操作

5.2 分类

例子

5.2 分类

例子

- Student 类

5.2 分类

例子

- `Student` 类
  - 小学生

5.2 分类

例子

- Student 类
  - 小学生
  - 中学生

5.2 分类

例子

- Student 类
  - 小学生
  - 中学生
  - 大学生

5.2 分类

例子

- Student 类
  - 小学生
  - 中学生
  - 大学生
    - 不同专业

5.2 分类

例子

- Student 类
  - 小学生
  - 中学生
  - 大学生
    - 不同专业
    - 不同学院

5.2 分类

例子

- Student 类
  - 小学生
  - 中学生
  - 大学生
    - 不同专业
    - 不同学院
    - 不同年级

5.3 设计和效率

5.3 设计和效率

- 软件效率分运行效率和生产(开发)效率

5.3 设计和效率

- 软件效率分运行效率和生产(开发)效率
- 运行效率包括运行时间耗用和空间耗用，关乎程序设计策略，数据结构，存储管理和算法(代码优化)

5.3 设计和效率

- 软件效率分运行效率和生产(开发)效率
- 运行效率包括运行时间耗用和空间耗用，关乎程序设计策略，数据结构，存储管理和算法(代码优化)
- 设计效率关乎设计方法和管理机制

5.3 设计和效率

- 软件效率分运行效率和生产(开发)效率
- 运行效率包括运行时间耗用和空间耗用，关乎程序设计策略，数据结构，存储管理和算法(代码优化)
- 设计效率关乎设计方法和管理机制
- 设计方法即程序设计方法，分层与抽象的角度看问题有利于理清数据处理的思路

5.3 设计和效率

- 软件效率分运行效率和生产(开发)效率
- 运行效率包括运行时间耗用和空间耗用，关乎程序设计策略，数据结构，存储管理和算法(代码优化)
- 设计效率关乎设计方法和管理机制
- 设计方法即程序设计方法，分层与抽象的角度看问题有利于理清数据处理的思路
- 面向对象程序设计方法能简捷地利用语言来表达分层与抽象(开发效率)，C++语言本身的类机制能很好地发挥程序的运行效率

5.4 再次讨论 JOSEPHUS 问题

5.4.1 过程化方法

一种偏向于函数式的思路：

将问题抽象为一般的数据结构，再进行处理

`example/lec05/josephus1`

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

如果只考虑最后一个：

```
int josephus(int n, int k) {
    if (n == 1)
        return 0;
    int r = josephus(n - 1, k);
    return (k + r) % n;
}
```

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

如果只考虑最后一个：

```
int josephus(int n, int k) {
    return n == 1 ? 0 : (k + josephus(n - 1, k)) % n;
}
```

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

$$f(n,k) = (k + f(n-1,k)) \bmod n$$

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

$$f(n,k) = (k + f(n-1,k)) \bmod n$$
$$f(1,k) = 0$$

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

$$f(n,k) = (k + f(n-1,k)) \bmod n$$
$$f(1,k) = 0$$
$$f(2,k) = (k + f(1,k)) \bmod 2 = (k+0) \bmod 2$$

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

$$f(n,k) = (k + f(n-1,k)) \bmod n$$
$$f(1,k) = 0$$
$$f(2,k) = (k + f(1,k)) \bmod 2 = (k+0) \bmod 2$$
$$f(3,k) = (k + f(2,k)) \bmod 3 = ((k+0) \bmod 2 + k) \bmod 3$$

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

$$f(n,k) = (k + f(n-1,k)) \bmod n$$
$$f(1,k) = 0$$
$$f(2,k) = (k + f(1,k)) \bmod 2 = (k+0) \bmod 2$$
$$f(3,k) = (k + f(2,k)) \bmod 3 = ((k+0) \bmod 2 + k) \bmod 3$$
$$f(n,k) = (((k+0) \bmod 2 + k) \bmod 3 + k \cdots) \bmod n$$

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

$$f(n,k) = (((k+0) \bmod 2 + k) \bmod 3 + k \cdots) \bmod n$$

```
int josephus(int n, int k) {
    int r = 0;
    for (int i = 2; i ≤ n; i++) {
        r = (r + k) % i;
    }
    return r + 1;
}
```

5.4 再次讨论 JOSEPHUS 问题

5.4.2 函数式方法

扩展阅读:纯函数式的几种方案（Haskell 版本）

```haskell
josephus n k =
  let loop xs = let d:r = drop (k-1) xs
                in d : loop (filter (/= d) r)
  in take n (loop (cycle [1..n]))
```

```haskell
josephus 1 k = 0
josephus n k = (k + josephus (n - 1) k) `mod` n
```

```haskell
josephus n k = 1 + foldl go 0 [2..n]
 where
  go r i = (r + k) `mod` i
```

5.4 再次讨论 JOSEPHUS 问题

5.4.3 面向对象方法

`example/lec04/joseph`

5.4 再次讨论 JOSEPHUS 问题

扩展阅读： APL 语言

Uiua 语言[1]版本

```
Joseph ← ○:○:⍥(⊙:⊂⊙(⊃(⊢|↘1)⍜↻)):[]⊙::⍜⧻+1↟⊙(-1)
Joseph 41 2
```

1. https://www.uiua.org

5.5 其他案例

raylib 相关案例

作业

用 `raylib`，借助弹簧函数：$x''(t) = -kx(t)$，实现多个小球连成链状的动画效果（每个小球都以前一个小球为弹簧中心，第一个小球以鼠标为弹簧中心，每个小球考虑重力，效果参考：https://www.bilibili.com/video/BV1zz411e7YN）