Java 编程与算法实用入门
A Concise and Practical Introduction to Programming Algorithms in Java

- **Chapter 8: Data-structures and object methods**
- 第 8 章：数据结构和对象的方法

Agenda 大纲

- FIFO data-structures 先进先出（ FIFO) 数据结构
- (= First In First Out)
- ● Heap data-structures 堆结构
- ● Non-static methods 非静态方法
- (= object methods) （ = 对象的方法）
- ● Revisiting lists (OO-style) 重访问链表（面向对象的类型）

FIFOs: Mastering queues 先进先出：控制队列

- Objects are considered in turn, one by one 对象轮流一个一个处理
- Process each object according to their arrival time 根据对象的到达时间来进行处理
- While objects are processed, others are **queued 当有对象被处理时，其它对象排队等待**
- First object should be first served! 第一个到达的对象被第一个处理

Basic examples: 简单的几个例子：

- Waiting in a queue at the post office. 在邮局排队
- Printing ?jobs> on a printer. 有多个文件同时需要打印机打印

FIFO = First In First Out ! FIFO = 先进先出 !

A basic solution 基本思路

- Stack objects in an array as soon as they arrive 一收到对象就把它存在栈中
- To stack an incoming object, should know the **index** of the last location
  要把对象存在栈中，就需要知道存储的位置（索引）
  - Should also know the **index** of the last processed object
    (so that we can process the next one)
    While processing an object, others can *come in* the array
    (= **queued**)

同时还需要知道上一个处理的对象的索引（这样才能知道下一个对象是哪个）。当我们在处理一个对象时，其它对象可以在数组中排队。

A basic solution 基本思路

- An array: container array for storing objects 一个 array(数组) ： 存储数组，用以存储对象
- **Two indices**: lastProcessed and freePlace 两个索引： lastProcessed （用以记录最后处理的对象位置）和 freePlace （用以记录可用空间的位置）
- To add an object x, we do array[freePlace]=x and we then increment: freePlace++ 增加一个对象 x ： array[freePlace] = x
- To process an object, we increment lastProcessed and we process array[lastProcessed]

当我们需要处理一个对象时，把 lastProcessed 加 1 ，然后我们取出 array[lastProcessed] 进行处理

Visual depiction of a FIFO 借助图来理解先进先出算法（ FIFO ）

FIFO: Queuing objects 先进先出（ FIFO ）：对象排队

Queuing another object 如何对新对象列入队伍

Processing and queuing 处理对象和把新对象列入队列

Processing and queuing can be done in parallel using threads 处理对象和把新对象列入队列可以用多线程并行处理

Programming queues 队列的代码

FIFO: First In First Out

FIFO ： 先进先出

Exercise: FIFO in action! 练习： FIFO

Let A be a set of integers such that: 设 A 为一个整数的集合，满足以下条件：

- 1 belongs to A, and 1 属于 A
- If a belongs to A, then 2*a+1 and 3*a belongs to A 如果 a 属于 A ，那么 2*a+1 和 3*a 都属于 A

Question: 问题：

For a given n, display all integers less or equal to n that belong to A.

任意给定一个 n, 显示出所有小于等于 n 并且属于 A 的数字。

Programming queues 给队列编程

Start with a FIFO initialized with element 1 首先用 1 给 FIFO 初始化

Use a boolean array to store whether a belong to A 构造一个布尔 (Boolean) 数组，用以记录每一个数是否属于 A

(= marks, tag elements) （相当于标签）

For each element a of the FIFO do: 对于 FIFO 的每一个元素：

- Compute 2*a+1 and 3*a 计算 2*a+1 和 3*a
- Add them to the FIFO if they are less than n...and not yet encountered (=marked) 如果他们小于 n 并且是一个新数字（没有被标记过）的话，把他们放入 FIFO 队列中

Terminate when the FIFO is empty 直到当 FIFO 队列为空时停止

Display all marked elements (=result) 显示所有被标记过的数字（结果）

A few remarks on FIFOs 关于 FIFO 的几点值得注意的地方

- Set beforehand the size of the array? 是否要事先确定数组的大小？
- Can wrap the array using mod MAX_SIZE 可以用 mod MAX_SIZE 来构造数组的功能

(=circular ring, extend arrays, etc.) （比如用循环的环，扩展数组等数据结构）

...But how to check whether the queue is empty 但问题是如何检验队列是否为空？

... or full with circular arrrays? 以及如何检验循环结构的数组是否已满？

Priority queues: Heaps (=tas) 优先队列（堆）

- Objects are considered in turn 对象被轮流考虑（等待处理）
- Need to process them according to their priorities 根据他们的优先级来处理
- While processing an objects, other may arrive 当处理一个对象时，其它新对象可能会到达（进入队列）

(= are being queued)

- Serve the object with the highest priority first 优先处理优先级最高的对象

Examples: 例子：

- Ressource request 资源请求
- Operating system tasks 操作系统的任务

Defining mathematically heaps 用数学语言来定义堆

A heap is a sequence of integers: 堆是一个被紧密存储在数组中的整数序列

stored compactly in an array such that:

For example: 比如：

37, 22, 31, 16, 17, 2, 23, 12, 6, 9

(heap of 10 elements) （ 10 个元素的堆）

Drawing a heap 画出堆

- Draw a heap as a tree (=special graph Vertex/Edge) 用树状结构（相当于特殊的图）来画堆
- Each node i contains a value t[i] and has 每一个节点 i 拥有一个值 t[i] 并且拥有 0， 1，或 2 个对应值小于他们父亲的兄弟
  0, 1 or 2 siblings that contain nodes of values less than its parent
  - Node i has children 2i and 2i+1 节点 i 有 2 个儿子： 2i 和 2i+1

37, 22, 31, 16, 17, 2, 23, 12, 6, 9:
Read layer by layer, 从树根到叶子，一层一层地读
from the root til the leaves

Storing and manipulating heaps 对堆进行存储和操作
Easy to code with a linear array 用线性数组很容易

Fundamental property of heaps 堆的基本性质
Largest value is stored at the root of the tree, namely 最大值被存储在树根，即数组的第一个位置
at the first element of the array.

Adding an element in a heap 在堆中增加一个元素

- Add the new element in position n (=n+1th element)... 把新元素放在位置 n (第 n+1 个元素）
- But the condition that the array is a heap is violated... 但这样的话数组就不是以堆顺序存储
- So that we swap the element until...

. ...it satisfies the heap constraint 所以我们需要改变元素的顺序（交换元素），直到满足堆的条件为止

Example: Add element 25 例子：增加元素 25

Not a heap anymore!!! 25>17 因为 25〉 17，这不再是一个堆结构了！！！

Swap with your parent! 交换父节点

Add element 25... and swap!!! 增加元素 25， 然后交换！！！
It is a heap now! 现在它是一个堆了！

Adding an element in the heap 在堆结构中增加元素

Removing the largest element of a heap 如何去除堆中最大的元素

- We move the element at position (n-1) and

put it at the root (position 0) 我们把在 n-1 位置上的元素放到根节点（位置 0）上

- But it is not anymore a heap... 但这是就不满足堆的性质了
- So we swap to the bottom until...

...the heap condition is satisfied 因此我们把元素交换到堆的底部，直到满足堆的条件

Removing the largest element of a heap 如何去除堆中最大的元素

Then Swap parent-child... 然后交换父子元素

Removing the largest element of a heap 如何去除堆中最大的元素

Non-static methods and objects 非静态方法和对象

- Do not write static in front of the method 不要在方法前使用 static 关键词
- Method is thus attached to the object for which it applies for 这样的话该方法就和他所应用的方法相关联
- For example, we prefer: 比如，我们偏爱

u.display() rather than display(u) u.display() 而不是 display(u)

u.addElement(a) instead of addElement(a,u) u.addElement(a) 而不是 addElement(a.u)

- To reference the object on which the method is called upon

use this 如果要表示引用该方法的对象，用 this 关键词。

Object-oriented programming paradigm (OO) 面向对象（ OO ）编程范例

Object-oriented programming paradigm (OO) 面向对象（ OO ）编程范例

- Design a software as a set of objects and methods applying on these objects 设计一个软件时，把软件视为一组对象以及一系列应用在这些对象上的方法。
- Ask yourself first: 首先问自己：

- What are the objects? 什么是对象？
- What are the methods? 什么是方法？

- Usually, a method often modifies the object (=fields) on which it applies for. 通常来说，一个方法能够修改它所应用的对象 (But not always, for example: Obj.Display()) （但也并非绝对，比如说 Obj.Display() 不能修改它所应用的对象）。

Picture ：

Public procedures 公共 (public) 方法
Private data 私有 (private) 数据
Private procedures 私有 (private) 方法
Interaction-Interface 交互 - 接口（ Interface ）

OO style: 面向对象（ OO ）风格：
object methods 对象方法？
versus 还是
static functions 静态函数？

Static methods are useful for defining: 静态方法用于在函数库（ library ）中定义：

- Constants 常数
- Basic functions 基本函数

....in a library. 等等。

Heaps revisited in Object-Oriented style 用面向对象（ OO ）风格编写的重访问堆

Observe that the keyword static has disappeared 我们不再使用关键字 static 了

List in object-oriented style 面向对象风格的链表（ list ）

- A cell stores information (say, an integer) and point/refer to the next one. 每一个单位存储信息（比如一个整数）和指向下一个单位的指针。
- Pointing to another cell means storing a reference to the corresponding cell. 指向下一个单位意味着存储对应单位的引用。

Pay special attention to null !!! 要特别注意 null!!!

- Remember that we cannot access fields of the null object 要记住，我们无法进入空对象的域
  - Throw the exception nullPointerException 会抛出异常 nullPointerException
- Thus we need to check whether the current object is null or not, before calling the method 因此我们需要在调用方法前检查对象是否为空

- In the reminder, we consider that all lists (also the void list) contain a first cell that stores no information. 在之后的课程中，我们假设所有的链表（包括空链表）的第一个不存储任何信息的元素。

Revisiting the linked list (OO style) 重新来看链表（ linked list ） (面向对象风格)

Revisiting the linked list (OO style) 重新来看链表（ linked list ） (面向对象风格)

Revisiting the linked list (OO style) 重新来看链表（ linked list ） (面向对象风格)

Revisiting the linked list (OO style) 重新来看链表（ linked list ） (面向对象风格)

Revisiting the linked list (OO style) 重新来看链表（ linked list ） (面向对象风格)

Stacks (LIFO): Last In First Out 栈数据结构：后进先出（ LIFO, Last In First Out ）

Two basic operations for that data-structure: 2 个基本操作：

- Push: Add an element on top of the stack 压入（ push ）： 在栈顶加入一个元素
- Pull: Remove the topmost element 弹出（ Pull ）：从栈顶移去一个元素

Stacks (LIFO) using arrays　用数组来实现栈（ LIFO ）

Stacks (LIFO) using linked lists　用链表来实现栈（ LIFO ）

Stacks: API　栈的应用程序接口（ API ）

// Use a Java package here　// 这里我们使用 Java 包

Stacks (LIFO) using linked lists　用链表来实现栈（ LIFO ）

Notice: Same code as StackArray demo program.　注意：与 StackArray 演示程序代码相同。

Static functions versus methods 静态函数还是对象方法

- Static (class) functions: 静态函数：

Access static/local variables only. 只能使用静态 / 本地的变量

《 class methods » 《类方法》

Potentially many arguments in functions 静态函数中容易产生许多参数

- Object methods: 对象方法：

Access object fields (using this) 可以使用对象的域变量或方法（用 this 关键字）

Access (class) static variables too. 也可以使用静态变量。

Objects are instances of classes 对象是类 (class) 的实例

Data encapsulation 数据包装

(=functions with limited number of arguments) （＝拥有少量参数的函数）

Constructor (= field initialization) 构造函数（＝域初始化）