**简介**

这一部分主要是实现SimpleDB对数据的管理，当然，需要在指导下先搭建好开发环境并了解SimpleDB的整体框架。更具体来说，需要实现存储、访问与管理物理层面的数据（二进制文件），以及将其映射为逻辑层面的数据（关系表）。在这一课题的最后，还要求实现SimpleDB中最基本的操作——SeqScan，因此完成这一章后，就可以扫描全表了。

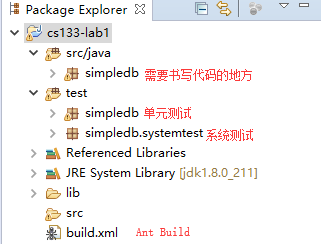
**目标**

* 实现管理元组的类，即*Tuple*，*TupleDesc*。已实现了*Field*，*IntField*，*StringField*和*Type*。由于只需要支持**整数**和**（固定长度）字符串字段**和固定长度元组。
* 实现*Catalog*。
* 实现*BufferPool*构造函数和*getPage*（）方法。
* 实现访问方法，*HeapPage*和*HeapFile*以及关联的ID类。这些文件的很大一部分已经编写。
* 实现运算符*SeqScan*。
* 通过*ScanTest*系统测试，这是本实验的目标。

**解压**

D:\Users\Ethan\workspace>tar -pzxvf cs133-lab1.tar.gz

**在eclipse中打开**



*图1 cs133-laba1 Package explorer*

对于完成的每个类，在test\simpledb\对应单元测试

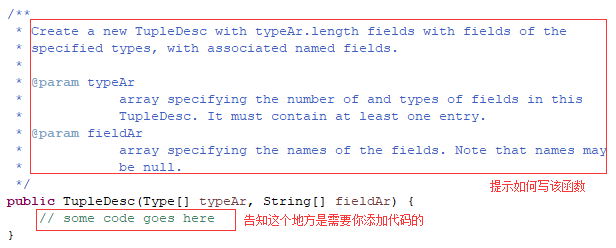
全部完成后，通过test\simpledb.systemtesst\ScanTest测试

关于Ant Build Targets?

：构建项目的工具，通过设置Target构建项目

**添加代码**

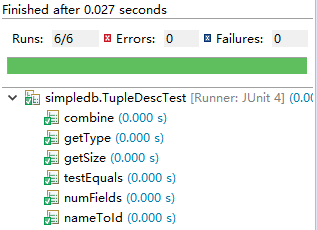
如图2，在遇到"//some code goes here"时，表明我们需要将该函数满足需求的填写完整，有时候为了达到目的，我们需要增加一些辅助的变量和函数来完成这些函数。此外，有的地方表明了"//not necessary for lab1"的字眼，表明我们在实验1中无需完成。函数定义上方有参数说明函数作用说明以及返回值说明等。



*图2 根据提示添加代码*

**测试**

如图3，在完成某目标中的某一点时，我们可以对当前完成的这一个类进行单元测试。测试的内容在展开的菜单中，比如对于TupleDesc的单元测试，有6个测试用例，分别为combine、getType、getSize、testEquals、numFields、nameTold。通过该测试有助于检查当前代码的正确性、完整性、在每个局部目标完成时我们都应该测试来检查一下，最后在完成整个实验1时通过系统测试SeqTest。

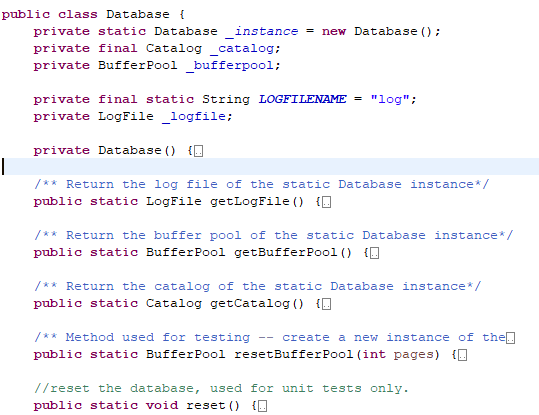


*图3 正确的测试结果*

**实验内容**

**1 Database类**

在Database.java中查看一下我们之后可能要使用的静态对象、方法的源码。



*图4 Database.java*

**2 Fields和Tuples**

**2.1 TupleDesc**

描述一张关系数据库表的「模式」（schema）。图5是TupeDesc的一种构造方法，其中typeAr数组存储「类型」（type）、fieldAr数组存储「字段名」（field name）除了几个构造方法，我们还要完成诸如numFields、getFieldName、getFieldType等等的方法。

TupleDesc支持用 index 获得字段的 type 或 name，用 name 获得字段的 index，还提供一个静态方法用于 merge 两个 TupleDesc 获得一个新的 TupleDesc （供Join operator 使用）

https://uploader.shimo.im/f/Q04yGq5tCPUB2bag.png!thumbnail

*图5 TupleDesc的一种构造方法*

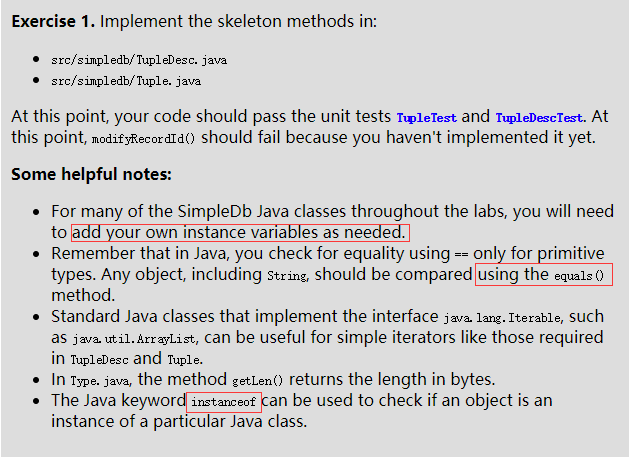
根据函数详细的基本要求，实现TupleDesc.java

**2.2 Tuple**

Tuple 存储字段，其除了提供第 i 个 字段的 getter/setter，还提供了所有字段的迭代器。Tuple 有一个RecordId标志其在磁盘中的位置。

值得注意的是，Tuple.java建立在补全完RecordId.java和TupleDesc.java的基础上完成。

**2.3 Exercise 1**



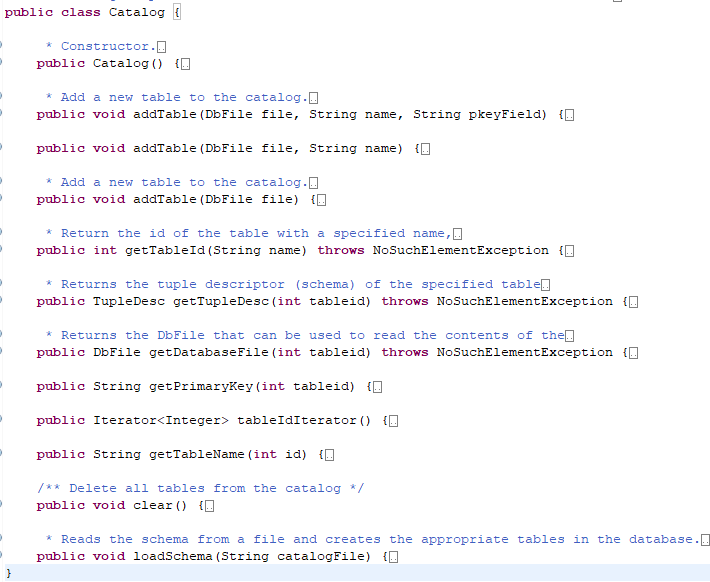
**3 Catalog**

**3.1 概述**

Catalog 储存了所有表的信息。每个表的信息包括：name，schema，相应的 DbFile，以及 primary key。

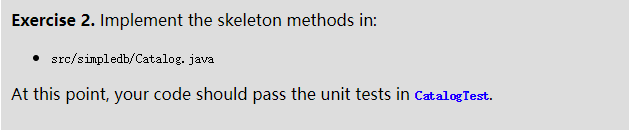
如图6，可以看出我们还需补全一些得到这张表相应信息的方法，例如getTableId、getPrimaryKey等方法，同时我们可以看到，添加表的方法有三个重载函数需要实现。

同时我们可以联系到第1节的\_Catalog静态变量。



*图6 Catalog方法*

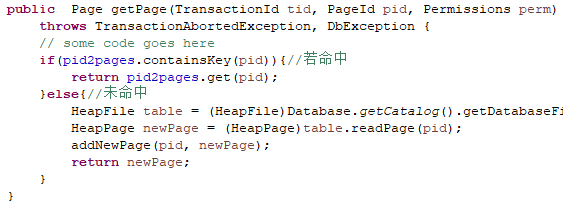
**3.2 Exercise 2**



**4 BufferPool**

**4.1 缓冲池的作用**

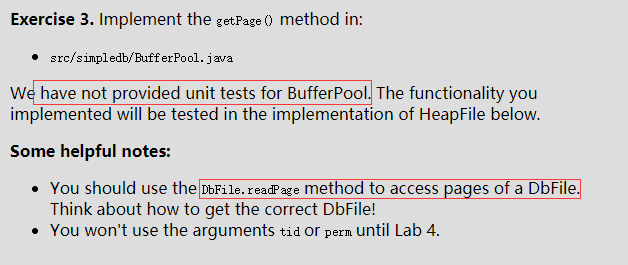
数据是储存在磁盘中的（支持序列化），当需要访问时，都会通过「缓冲池」获得。缓冲池调用相应的 HeapFile 从磁盘中获得数据并生成 HeapPage 实例放入缓冲池。当缓冲池已满，会淘汰一个页面，如果那个页面是脏页面的，会先 flush 到磁盘（通过调用 HeapFile 的 writePage API）。



*图7 tid事务访问某页面*

在本实验中，只需要实现SeqScan运算符使用的构造函数和BufferPool.getPage（）方法。

**4.2 Exercise 3**



**5 HeapFile访问方法**

**5.1 堆文件（HeapFile）**

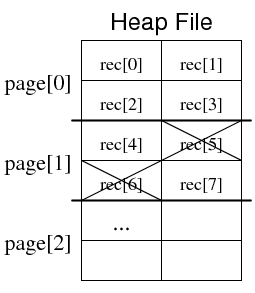
访问方法能从磁盘读写数据（特定方式排列）。常见的访问方式包括

* 堆文件「我们只要实现这种」
* B-Tree

每张表有一个HeapFile对象。

HeapFile对象是一组页面，每个页面由固定数量（BufferPool.PAGE\_SIZE）字节组成。

HeapFile中的每个页面都被设为一组插槽，其中每个插槽可以容纳一个元组（SimpleDB中给定表的元组都具有相同的大小）。



*图8 HeapFile Structure*

图8并不完全符合SimpleDb，因为除了这些插槽之外，每个页面都有一个header，该header用一个bit记录该页面中每个插槽有效情况，形成bitmap。1表示有效，0表示无效（例如已删除或从未初始化）。比如1010表示“1有效2无效3有效4无效”这样一个bitmap。

**5.2 堆页面（HeapPage）**

5.1节中的「页面」其实为「HeapPage」。HeapFile对象的页面是HeapPage类型，它实现了Page接口。页面存储在缓冲池中，但由HeapFile类读取和写入。它支持在该页面上插入/删除元组，标志该页面为 dirty。还提供了迭代器用来迭代页面中所有元组。

因为 tuple size、page size 单位是字节，所以需要 \*8 ，1bit用于header。因此，某页面中的元组数量是：

*tuples per page* = floor((*page size* \* 8) / (*tuple size* \* 8 + 1))

* floor（）是指向下舍入得到整数。

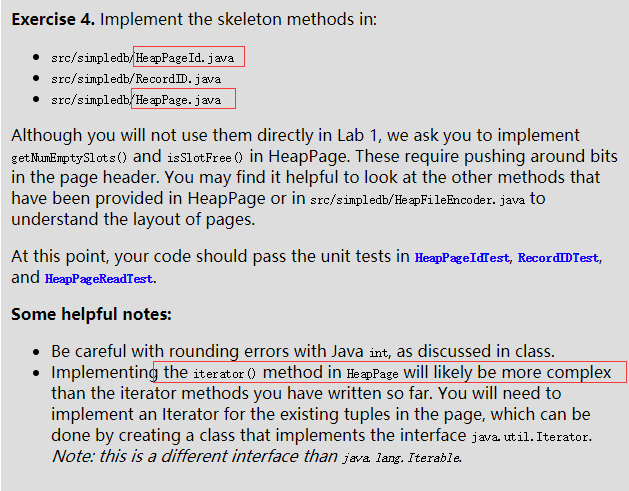
根据每页的元组数，存储header所需的字节数就是：

headerBytes = ceiling(tupsPerPage/8)

* ceiling（）是指向上舍入得到整数

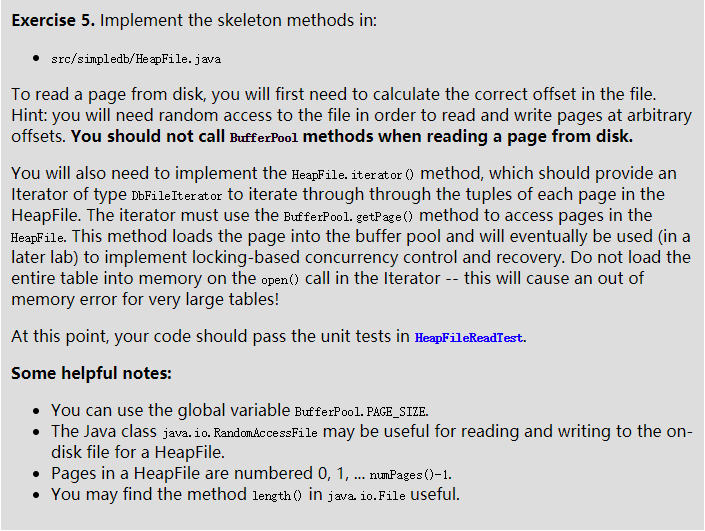
介绍完了基础概念有助于接下来的编程练习。

**5.3 Exercise 4**



**5.4 Exercise 5**

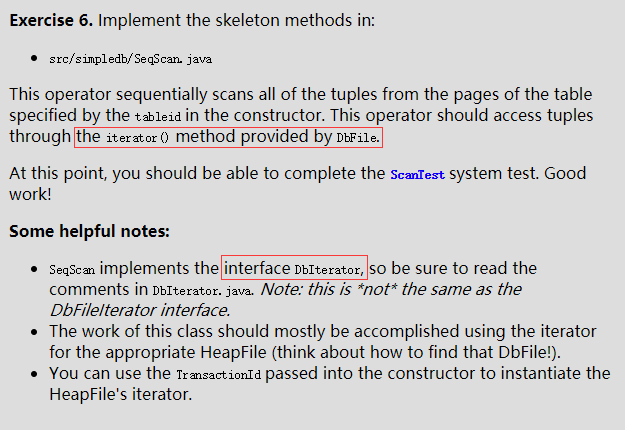
实现了HeapPage后，在本实验中编写HeapFile的方法来计算文件中的页数并从文件中读取页面。 然后，我们将能够从存储在磁盘上的文件中获取元组。



**6 Operator**

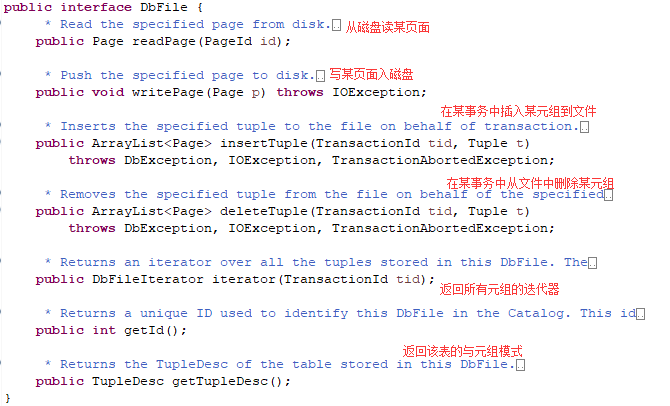
Operator 就是迭代器的连接，它实现 DbIterator 接口，接受 child DbIterator。完成SeqScan.java，实验1也就全部完成。通过系统测试ScanTest以检测正确性。

**6.1 Exercise 6**



**7 其他**

DbFile：每个DbFIle代表一张表，成员方法如图9。



*图9 DbFile*