CSE Lab3说明文档

PartA

汉明距离计算函数

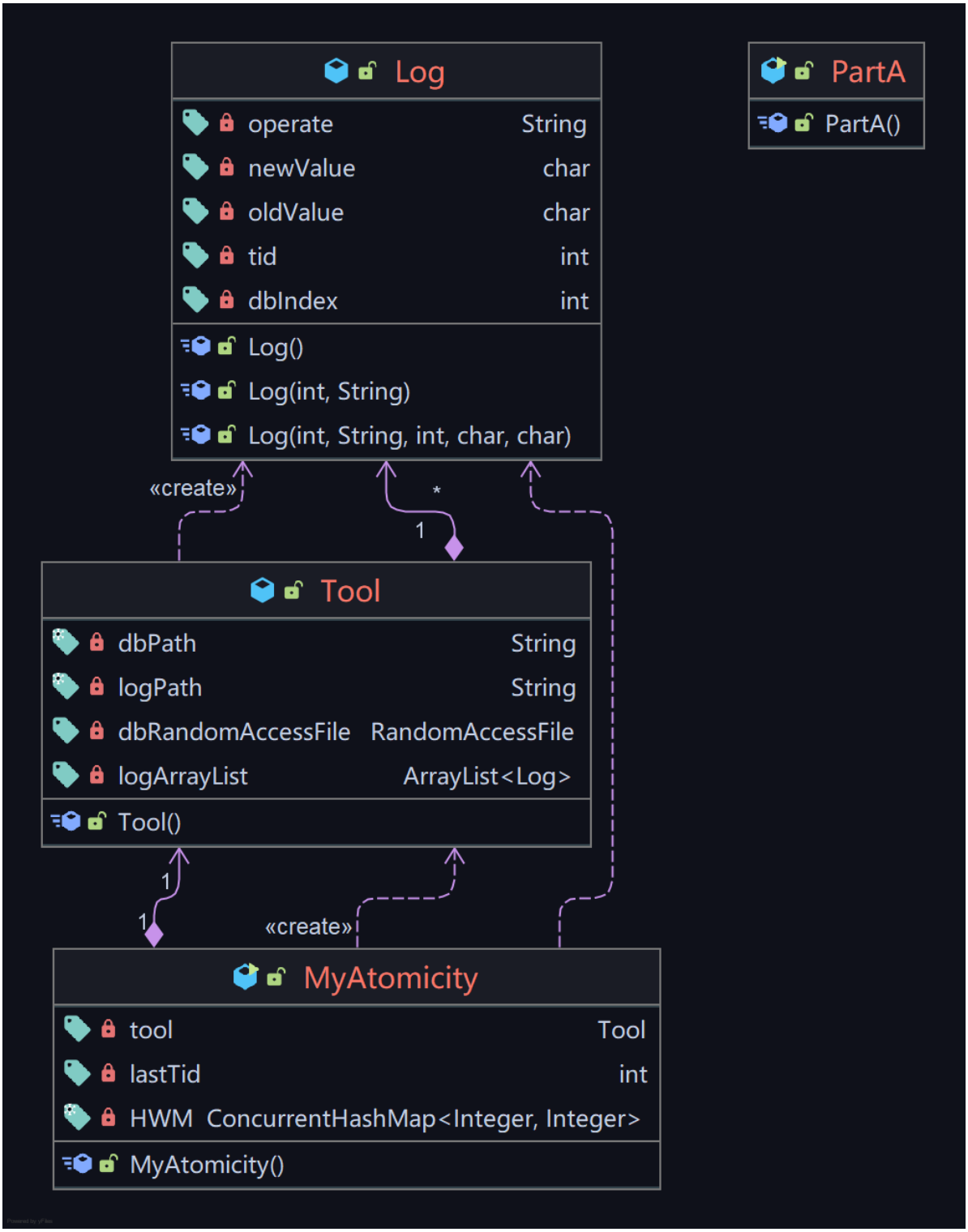
```
public int hammingDistance(int x, int y) {
    int hamming = x ^ y;
    int sum = 0;
    while (hamming!=0){
        sum += hamming & 1;
        hamming = hamming>>1;
    }
    return sum;
}
```

计算机系统中的可靠性

计算机系统可靠性指在规定条件下和给定时间内计算机系统正确运行的能力或可能性。在实际的运行环境中，计算机系统可能会因为自己本身的随机性故障、不良的运行环境或者其他的干扰因素而不能正常运行。比如黑客，他们就通过使用各种攻击手段和错误程序对计算机系统进行攻击，从而降低计算机系统的的稳定性以谋取利益。针对这种情况计算机系统可以对外部不良信息和恶意软件进行拦截，从而大概率避免被计算机病毒攻击，提高了计算机系统运行的安全性与可靠性。

PartB

代码结构




```java
//单个事务
public class MyAtomicity{
    // 该方法启动一个线程完成整个文件的更新操作
    public void update(char ch) throws IOException, InterruptedException {

    }
    // 该方法通过反向读取日志文件完成恢复操作
    private synchronized void recover() throws IOException {

    }
    // 当某个线程要修改文件的某行时，检查该行数据的HWM是否小于线程编号，小于则数据能用，大于不能用
    private boolean checkHWM(int tid,int dbIndex){
```

```
    }
    // 完成abort的事务的新线程的线程编号
    private synchronized int getNewTid() {

    }
}
```

```
// 自定义的日志类，日志组成：线程编号+执行的操作+处理的文件行数+旧值+新值
public class Log {

}
```

```
// 工具类
public class Tool {
    // 初始化，读取日志文件转换为自定义的log结构
    public Tool() throws IOException {

    }
    // 每个线程运行完保存日志
    public synchronized void saveLogs() throws IOException {

    }
    // 处理write，写进日志
    public Log doAffairs(String operate,int tid,int dbIndex,char oldValue,char
newValue){

    }
    // 处理start commit abort，写进日志
    public Log doAffairs(String operate,int tid){

    }
    // 按行读取db.txt
    public synchronized char readDB(int dbIndex) throws IOException {

    }
    // 按行写db.txt
    public synchronized void writeDB(int dbIndex, char newValue) throws
IOException {

    }
    // 通过日志查看上次运行结束创建的最新的线程编号，用于recover
    public int getLastTid(){

    }

}
```

设计决策

对单个事务

依靠log

1. 事务start，将start写入log，然后修改数据，如果没修改就中断了，不需要恢复，因为日志中没有write记录
2. 如果数据还未修改完就中断了，反读日志，根据日志的write记录还原数据直至log顶部或者遇到commit，最后的结果就是返回到上次修改成功后或者完全没修改之前的数据
3. 数据修改完后中断，程序开始时反读日志至commit这个过程中没有write记录，不需要修改，数据还是一致的

start说明事务开始执行了，commit说明事务执行成功了，write记录每次要修改的数据，每次程序开始运行前都recover，通过上面的步骤保证在运行前数据就是一致的

对多个事务

在单个事务的额外条件下需要保证在多个事务并发执行的条件下，每个事务执行后就像单个事务执行后一样，只有事务执行的先后顺序不同而已，在本例中就是要保证多个事务并发执行后文件每一行的数据要保证一致

如果每个线程按序执行，不会出错，但是就失去了并发的意义，因为一个线程在处理某个变量时不会用到其他的变量

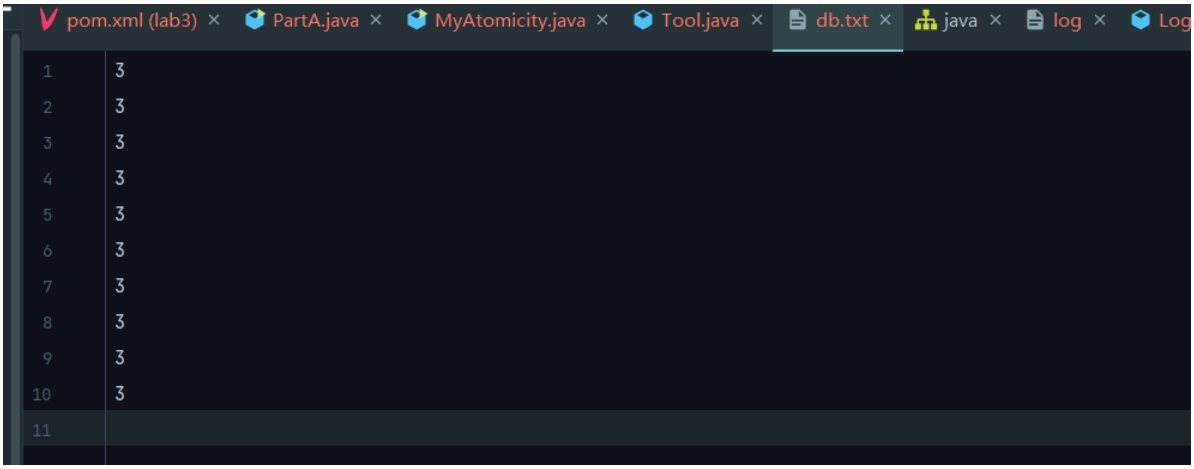
1. 这里使用read capture，给每个变量一个HWM初始值，每个线程要使用变量时先检查该变量的HWM是否小于自己给定的线程编号，小于就可以使用，大于就abort，然后重新开启一个线程表示一个事务来完成abort掉的事务需要完成的事
2. 如果运行过程中断，还是在每次开始时使用和单个事务一样的recover方法恢复数据

测试

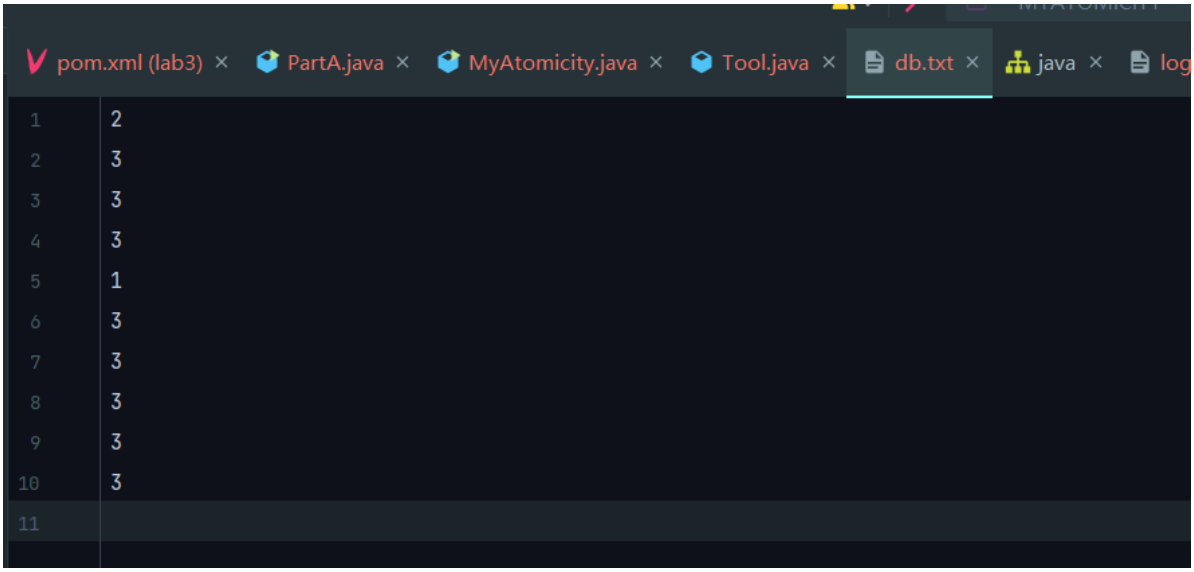
共开启3个线程表示三个事务

```
public static void main(String[] args) throws IOException, InterruptedException {
    MyAtomicity atomicity = new MyAtomicity();
    atomicity.update( ch: '1');
    atomicity.update( ch: '2');
    atomicity.update( ch: '3');
}
```

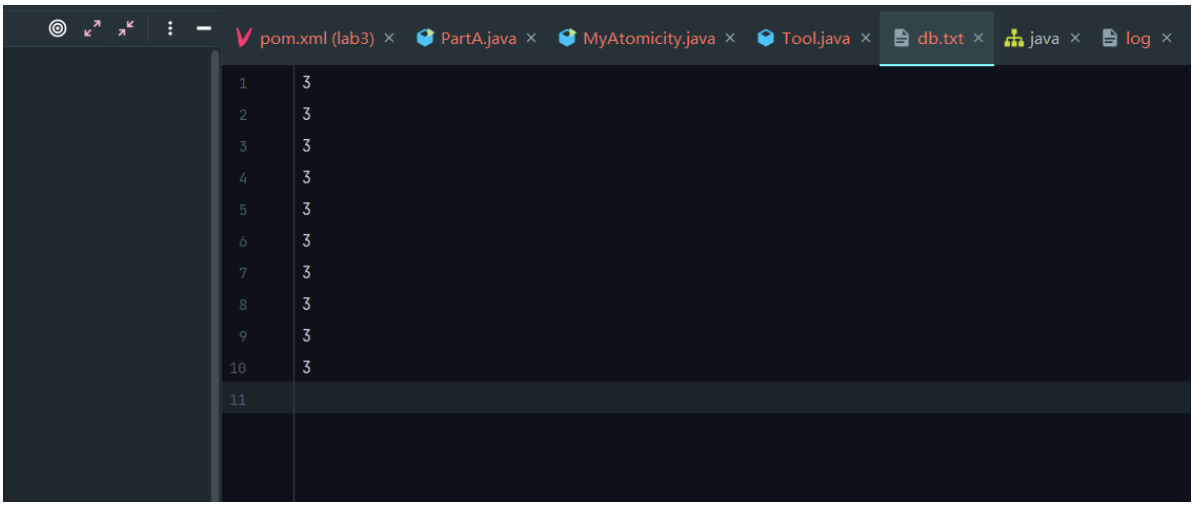
开始修改，修改之后的数据



手动中断修改来模拟事务意外中断



因为意外中断，出现了数据不一致，重新启动



成功恢复并修改