一、实验目的：

RSA 算法的实现

二、实验环境：

运行环境：eclipse

系统环境：Mac OS X 10.10

三、编写语言：

Java

四、实验内容：

该程序主要用于实现数据的 RSA 加密及解密，具体内容如下：

1. 输入密钥规模 N，程序自动生成两个大素数，并显示；
2. 输入一个公钥 e，程序自动生成一个私钥 d，并显示；
3. 可以对指定的 txt 文档进行加密或者脱密处理，并生成对应的密文或者明文 txt 文档。

五、实验步骤：

首先，考虑到产生的大数是 2048 位这种数量级的，任何的数值型都无法对其进行表示，我们必须自己定义一个基于字符数组的结构体。在查阅了相关资料后发现，Java 的标准库中就有这种已经封装好的类型可以直接使用—BigInteger。此外，还提供了 BigInteger 类型数据的加减乘除、大小比较、输出运算符、赋值、循环移位等。

做好这些准备工作之后，读取用户输入的位数信息，根据位数信息产生两个大素数符合 N = p*q。产生的过程具体如下：

1. 素数搜索

随机产生一个奇数，对以该数为起点的奇数依次递增 2 进行测试，直至找到一个素数。代码部分流程如下：

2. 素数筛选

准备工作：

费马小定理：对于素数 p 和任意整数 a，有 $a^p \equiv a(\text{mod } p)$（同余）。反过来，满足 $a^p \equiv a(\text{mod } p)$，p 也几乎一定是素数。

伪素数：如果 n 是一个正整数，如果存在和 n 互素的正整数 a 满足 $a^{n-1} \equiv 1(\text{mod } n)$，我们说 n 是基于 a 的伪素数。如果一个数是伪素数，那么它几乎肯定是素数。

Miller-Rabin 测试：不断选取不超过 n-1 的基 b(s 次)，计算是否每次都有 $b^{n-1} \equiv 1(\text{mod } n)$，若每次都成立则 n 是素数，否则为合数。

在 isPrime 函数中，我们采用 Miller Rabin 法进行筛选。根据 Miller Rabin 法，通过 r 次测试之后，错误概率不超过 1/4 的 r 次幂。伪代码如下：

```
Function Miller-Rabin (n : longint) :boolean;
begin
    for i := 1 to s do
    begin
        a := random(n - 2) + 2;
        if mod_exp(a, n-1, n) <> 1 then return false;
    end;
    return true;
end;
```

六、实验的难点

由于调用了 java 的 math.BigInteger 包中的方法，算法的每个步骤都可以直接实现，实验的稍难点在于文件的读写，我们需要通过使用 BufferedWriter 和 BufferedReader 进行实现

七、程序运行结果展示：

```
D:\javacode\RSA\bin>java RSA
输入密钥规模（位数）：
1024
p:751082930404771216334566393666295645110140861397241014439029061138769803683984
785194132286936590917046727679223096735492458734505046083533954837775479211 3
q:103078242952331387111352446556869768527713095550490585404533301382370150501642
329130990597020587543089655979043941449509020039896173241000684707572913450 93
```

```
输入公钥e：
65537
N:774203087776120143400039534286507820828910770614777109668933129005200639454193
044336177829593737549909329077525959918229680820722021154948521964192637530391 67
163048921871716322854061487280823328932967515702983289286484558600280884404745 97
593904348622901718373183252200876112057665007267428504970858035765150 9
私钥d:586526440149600456533133388426768288205981655568971474053779236997546603 42
860803288663242503210506054592350426726749683543117244988861777612822561552181 78
296158649579133443394498668756075658545556044288323745239829872803862702253929 97
185322852562347487518398775775101226636151543541793606086593814327741075 3
```

```
1.加密文件；2。解密文件；3.退出
1
输入明文文件：
message.txt
明文：7
加密后：169369533288220865583341661770587867326304573335685557742303734981995118
970540303625826846636058090967468496713442522965314585523780537930936471802271 91
710270552178610673154610809412548735164885200375450014681272087472297664157773 00
371166136001426491059485340374661738350159519594534748492678905269884988039 6
1.加密文件；2。解密文件；3.退出
2
输入密文文件：
verify.txt
密文：169369533288220865583341661770587867326304573335685557742303734981995118 97
054030362582684663605809096746849671344252296531458552378053793093647180227191 71
027055217861067315461080941254873516488520037545001468127208747229766415777300 37
116613600142649105948534037466173835015951959453474849267890526988498803 96
解密后：7
1.加密文件；2。解密文件；3.退出
```