CSE 11 Spring 2015 **START EARLY!**
Program Assignment #5 (100 points)
Due: 15 May 2015 at 1159pm (2359, Pacific Daylight Time)

PROGRAM #5 : Cryptor

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

The program focuses on using inheritance, abstract classes, handling simple exceptions, and handling command-line arguments. You will be creating a command-line java application that does very primitive encoding (encrypting) and decoding (decrypting). Please note that none of these algorithms should ever be thought of as good/safe/secure encryption algorithms.

There are three top-level classes: `Cryptor`, `StreamPair`, and `CryptStream`. `Cryptor` is the command line program. `StreamPair` creates an input/output pair of I/O streams, depending upon user input. `CryptStream` is an abstract class and is the superclass of the encoders/decoders. Your program will define three different encoders (and therefore subclasses of `CryptStream`) to do no encoding (plain), rotation encoding(rot13), and key-based XOR encoding (key).

We will begin by describing how `Cryptor`, the command-line application should work from a user's point-of-view, and then will describe the class hierarchy to guide your implementation.

1. Usage statement. If there is a detectable error or the user types "help" as the first argument, a usage statement should be printed, and the program should exit. eg. to get help

```
$ java Cryptor help
Cryptor [-d algorithm] [-e algorithm] [-k key] [-i infile] [-
o outfile]
     algorithms: plain rot13 key
```

2. Command line arguments. All command line arguments are *case-sensitive*. The order of arguments does not matter.

`-i`	*`infile`*	*`infile`* `is the name of the input file. If "-" is the name of the` *`infile`*`, then standard input (System.in) should be used. If this flag is not given, standard input is used`
`-o`	*`outfile`*	*`outfile`* `is the name of the output file. If "-" is the name of the` *`outfile`*`, then standard out (System.out) should be used. If this flag is not given, standard out is used`
`-d`	*`algorithm`*	`decrypt the input and write the decrypted stream to the output using the named algorithm. Algorithms supported are plain, rot13, and key`
`-e`	*`algorithm`*	`encrypt the input and write the encrypted stream to the output using the named algorithm. Algorithms supported are plain, rot13, and key`

-k	key	For the key algorithm, this is the key used for encrypting or decrypting the input stream

3. Giving the same argument multiple times is not an error. The last argument is the one to use. For example, `java Cryptor –i Cryptor.java -i StreamCrypt.java` will use StreamCrypt.java as the input file.

4. If neither `–d` nor `–e` are given on the command line, the <u>default is to encrypt using the plain</u> encryptor. This copies the input stream to the output stream.

5. Detectable errors:

   - Any File IO Errors encountered when opening your pair of streams
   - Any improper arguments including unsupported flags (not –d,-e,-k,-I,-o), asking for an unknown encryption/decryption algorithm

6. Examples Command line invocations

Copy the standard input to the standard output (if stdin is the keyboard ctrl-d will end the input)

```
java Cryptor
```

Copy the file Cryptor.java to standard output (same effect as `$ cat Cryptor.java`)

```
java Cryptor –i Cryptor.java
```

Copy the file Cryptor.java to standard output with rot13 encryption

```
java Cryptor –i Cryptor.java –e rot13
```

Encrypt file Cryptor.java save in Cryptor.enc using rot13 encryption

```
java Cryptor –i Cryptor.java –o Cryptor.enc –e rot13
```

Decrypt file Cryptor.enc write to standard output using rot13 encryption

```
java Cryptor –i Cryptor.enc –d rot13
```

Encrypt file Cryptor.java write to standard output using key encryption and key "Horace"

```
java Cryptor –i Cryptor.java –e key –k Horace
```

Encrypt the non-existent file Cryptor2.java write to standard output using key encryption and key "Horace"

```
$ java Cryptor -i Cryptor2.java -e key -k Horace
java.io.FileNotFoundException: Cryptor2.java (No such file or directory)
Cryptor [-d algorithm ] [-e algorithm ] [-k key] [-i infile] [-o outfile ]
         algorithms: plain rot13 key
```

Encrypt file Cryptor.java using non-existent algorithm key2 encryption and key "Horace"

```
$ $ java Cryptor -i Cryptor.java -e key2 -k Horace
Unknown algorithm: 'key2'
Cryptor [-d algorithm ] [-e algorithm ] [-k key] [-i infile] [-o outfile ]
         algorithms: plain rot13 key
```

Encrypt file Cryptor.java using algorithm key encryption and key "Horace", but output file is not writable

```
$ java Cryptor -i Cryptor.java -o /Cryptor.enc -e key -k Horace
java.io.FileNotFoundException: /Cryptor.enc (Permission denied)
Cryptor [-d algorithm ] [-e algorithm ] [-k key] [-i infile] [-o outfile ]
         algorithms: plain rot13 key
```

Encrypt file Cryptor.java using algorithm key encryption and key "Horace", but you have an unknown command flag

```
$ java Cryptor -i Cryptor.java -o Cryptor.enc -e key -k Horace -j arrow
Unknown flag '-j'
Cryptor [-d algorithm ] [-e algorithm ] [-k key] [-i infile] [-o outfile ]
         algorithms: plain rot13 key
```

You are not supplied any version of `Cryptor.java`

Class StreamPair

The `StreamPair` class is straightforward. Its constructor is given an input file name and an output file name. It then creates an object that holds references to the appropriate input and output streams. Note in the documentation the special meaning of "-" when used as the name for input file or output file. It also supports a `close()` method that will close the streams that are files. You are given starter code and you should use it to generate the Javadoc for it.

Class CryptStream

CryptStream is abstract class with a protected field, two abstract methods (which must be made concrete in the subclasses: `PlainCrypt.java, Rot13Crypt.java, KeyCrypt.java`). You will also notice two `final public` methods `encode()/decode()`.

Part of this assignment is for you to turn the following English statements into appropriate code and classes. It might help to draw on some paper how different classes relate to one another. <u>The first is the high-level logic of Cryptor,</u>

1. Read Command-line arguments
2. Create a `StreamPair` instance based upon the names of the input/output files
3. Using the `StreamPair` instance, construct a `CryptStream` instance based upon the chosen encryption/decryption algorithm. If algorithm is plain, then construct a `PlainCrypt` Instance, if algorithm is rot13 construct a `Rot13Crypt` instance, and if the algorithm is key, construct a `KeyCrypt` instance.
4. If encrypting, invoke encrypt() on your `CryptStream` Instance
5. If decrypting, invoke decrypt() on your `CryptStream` instance

6. close the `StreamPair,` by invoking `close()` on your StreamPair instance

The second is the logic of encrypt/decrypt in `CryptStream`

1. encrypt() should utilize the (abstract) method cryptData() to perform the actual encryption
2. decrypt() should utilize the (abstract) method decryptData() to perform the actual encryption
3. Notice that the arguments to cryptData and decryptData are byte arrays. Look up InputStream and OutputStream to see how to read/write byte arrays.
4. Also notice that you are dealing with streams, you don't know and cannot find out how much data is in the stream, until you read it. You will have to read the input stream in "chunks" until there is no more input left to read. Reading/writing streams are performed in encrypt() and decrypt().
5. To read in chunks of bytes, you will want to define fixed-sized buffer (of type `byte[]`), then you can use the InputStream's method `read(buffer)`. That `read` method will tell you how many bytes were actually read into your buffer. You should look at the documentation of this method to see how End-of-File (meaning the stream has no more bytes) is returned.
6. `encrypt()` and `decrypt()` are declared final. And they are the only public methods (other than the constructor) available to you. This means the Cryptor will invoke encrypt()/decrypt() on a specific `CryptStream` instance. It also means that basic algorithm for encrypting (or decrypting) data is coded in `CryptStream` using the abstract methods `encryptData()` and `decryptData()`. The concrete subclasses of `Cryptstream` implement working `encryptData()/decryptData()` methods.

When you Implement CryptStream, **You may NOT**

1. Change the signature of any public method
2. Add a "throws" clause to any public method
3. Add any new public or protected methods, you may (and probably will) add private methods
4. Add any public instance or class variables (you may add private instance/class variables)
5. Add any protected instance or class variables
6. Remove `abstract` or `final` from any of the method given method defintions

The concrete subclasses of `CryptStream: PlainCrypt, Rot13Crypt, KeyCrypt`.

Here we focus on just the encryption algorithms used in these files. These are implemented in the methods `encryptData` and `decryptData`. Note that these methods in these particular classes are not abstract and these classes are not abstract. Also, in these methods, the length of the input array may be longer than the supplied len argument. Below, a "plain" byte is the cleartext or non-encrypted byte. These should each be fairly short to code. Without comments, the professor's solution are 20 lines for PlainCrypt, 26 lines for Rot13Crypt, and 32 lines for KeyCrypt.

`PlainCrypt`:

Encryption/decryption are the same, each byte of the input byte array is copied to the output byte array.

`Rot13Crypt`:

For each byte: encrypted byte = (plain byte + 13) % 256
Decrypted byte = ?? (what's the inverse of the above operation?)

`KeyCrypt`

For each byte: encrypted byte = `key ^ plain` (recall that ^ is the XOR operator) byte
Decrypted byte = `key ^ encrypted byte`

We haven't done bit operations, so the above may not be obvious when using ^ the bitwise XOR operator.

Notes on `KeyCrypt`: it is the only class that uses a key. You will need to define a constructor that includes a key. E.g. `KeyCrypt(StreamPair theStreams, String Key)`

Now you have a problem with a key that is of type `String` and attempting to XOR with something of type byte. You need a way to "convert" a `String` key into a `byte`. One suggestion is to sum up all the bytes in the key string (In the `String` Class the method `getBytes()` will return the String as an array of bytes). Once you have summed all the bytes, you can cast the result into a byte-sized key. (that's not a very secure key! There are only 256 unique keys).

When you Implement `PlainCrypt, Rot13Crypt, KeyCrypt,`

1. **You may NOT** define any new public or protected methods.
2. You may add private methods.
3. You may add public constructors.
4. **You may NOT** Add any public instance or class variables
5. **You may NOT** Add any protected instance or class variables

You may (and must) create public constructors.

Hints. Here is a suggested order of how to develop your program. Developing in stages and testing each stage is a much faster way to develop code. Do not attempt to write all the code at once and then start debugging, develop in stages. It will also help you better understand how the classes relate to one another. Tutors are instructed to not attempt to help you debug the "whole thing at once". They will ask you about your stages of development, what works, what doesn't.

1. Implement `StreamPair`. It's the smallest and most straight forward code. Note that is should catch and re-throw exceptions if files cannot be opened (this can happen if the file doesn't exist on read, or if you do not have permission to read/create a particular file)
2. Write a first version of `Cryptor` that supports the "-i" and "-o" flags, creates a `StreamPair` instance and closes the stream pair. Check that errors (e.g., file not found, bad permissions) are properly handled and a nice usage statement is printed
3. Modify `Cryptor` to read the complete input stream and copy to the output stream using your `StreamPair` instance. At this point, you have only coded `StreamPair` and some of `Cryptor`. The code that reads/writes streams will eventually be moved into `CryptStream.`

4. Implement `CryptStream` and `PlainCrypt`. You need to do these together because you cannot create an instance of `CryptStream`. The idea here would be to Modify `Cryptor` to use a `PlainCrypt` instance to copy input to output.
5. Once `PlainCrypt` is working, you have all the "plumbing" in place. Modify `Cryptor` to support choosing multiple alogrithms (maybe just plain and rot13 at this stage). Then implement the rot13 encryptor. Make sure you can encrypt and decrypt with rot13.
6. Finally, implement the key encryptor and make appropriate changes to `Cryptor` to support it.
7. Test things top to bottom. Make sure that error conditions are still handled nicely. Verify that adding new code didn't break anything.

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundlePR5" program that will turn in the file
Cryptor.Java, StreamPair.java, CryptStream.java, PlainCrypt.java, Rot13Crypt.java, KeyCrypt.java

No other files will be turned in and they **must be named exactly as above.** BundlePR5 uses the department's standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

$ **/home/linux/ieng6/cs11s/public/bin/bundlePR5**

The output of the turnin should be similar to what you have seen in your previous programming assignments

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Don't forget to turn in your best version of the assignment.

Frequently asked questions

What if my programs don't compile? Can I get partial credit? No. The bundle program will not allow you to turn in a program that does not compile.

If only implement some of the encryption, can I get partial credit. Yes. But you need all files to even turn in. You should remark in comments that you know something is not implemented correctly.

If the user enters unknown flags, what should my program do? See sample output above.

If the user enters an odd number of arguments (eg. `$ java Cryptor –e`) , what should my program do? print the usage statement and exit. Note that only an even number of arguments can be passed to your program

Suppose somebody enters "java Cryptor –e rot13 –d rot13" what should my program do? whichever was the last one "wins". In this case, you should decrypt.

How about something like "java Cryptor –e plain –d rot13" ? It should decrypt using rot13. Think of the –e/-d flags as opposites, you are either encrypting or decrypting.

OK, how about "java Cryptor –e –i - rot13"? That's an error. This can be interpreted as "encrypt using the –i algorithm" followed by unknown flag "-"). You may register either error.

The examples above show the particular Exception generated. Do I have to do that? No, you can just print a usage statement (to System.out). But it's convenient for you to see exactly which Exception was generated. If you decide to add extra error output, please write it to `System.err.` If the exception caught in the catch block is `e`, then you can simply do `System.err.println(e)` to write a human-readable form of the exception.

Do I write the usage statement to standard output or standard error? Write it to standard output (System.out).

I can only see how to do this assignment if I add public method or add a throws clause to an existing method. Can I do that? No. Part of this assignment to understand how these classes can fit together to solve the problem at hand within the "restrictions".

Can I add a public constructor to anything besides PlainCrypt, Rot13Crypt and KeyCrypt? No.

I have to add public constructor(s) to PlainCrypt, Rot13Crypt and KeyCrypt, right? Yes.

Do I have to check for all kinds of crazy inputs? Everything entered on the command line is a String, you shouldn't be converting Strings to integers. Your program should gracefully handle file-related exceptions in `StreamPair`.

Can you give some sample input/output? Yes. See download site

Why is this so complicated? Actually, it's not. This kind of processing and software architecture is actually quite common. Once you are done, ask yourself "how difficult would it be to add another encryption algorithm?"

When I enter the same key to decode the sample key-encrypted input I get garbage? You may not be calculating the byte version of the string in exactly the same way the program used to generate file. Your version must be able successfully encrypt and decrypt with a given key.

Will you grade program style? Yes. In particular, indentation should be proper, variable names should be sensible. We will also look for code clarity, too. Overly long or complex codes are frowned upon. One to think about is in your implementation of CryptStream – how different are encoding and decoding? Would a helper method be of use? For reference the professor's full implementation of this project (including comments) is 326 lines.   You are being supplied about 94 lines of code and comment. Your code may be longer or shorter in both cases. You are encouraged to write Javadoc-style comments for all your methods, public and private. Writing Javadoc-style comments is not a requirement, but you should get in the habits of writing suitable comments.

What is a suitable comment? Comments help you, the programmer, understand the logic of your code. If some piece of logic was difficult to work out when coding, a note to yourself (in a comment) should remind you about the logic. Picture yourself picking up this code next quarter (after a long summer's break), would it still be obvious.

Should I comment every line of code? Absolutely NOT! A particularly bad/useless comment would be something like

```
     count = nelem;     // assign nelem to count
```

You simply restated in English, what the code is obviously doing.

Generally, one puts a comment at the top a block of code, say what the block does (or what it is supposed to do).   This allows a reader of your code to skip over the program detail.