# Chapter 1 - Introduction to java

Skip to end of metadata
Go to start of metadata
**Reading Material:**

Working environment

- Eclipse Tutorials
- Eclipse Shortcuts
- Eclipse Debugging
- Maven Tutorial

Head First – Java (Read all the listed chapters)

- 1 – Breaking the Surface
- 2 – Trip to Objectville
- 3 – Know Tour Variables
- 4 – How Objects Behave
- 5 – Extra-Strength Methods
- 6 – Using the Java Library

**Excercise:**

Open a new project in eclipse (or an other IDE) and call it "encryptor".

Input and output options to the user should be via the command line.

You should write an encryption program (Pay attention to the emphasis in the end of the exercise), and the program should work in the next manner:

1. Should allow the user to choose between Encryption and Decryption (build a convenient menu to the user).
2. In case of Encryption:
a. Inputs the path to the source file from the user.
b. Randomize a key for the encryption – some random number
c. Uses the key to encrypt the content of the file

The encryption method: Adding the value of the key to each character in the string.

1. The result should be written to a new file, and the name should be in the next format:
[original-file-name]_**encrypted**.[original file extension]
for example, if the source file name is "C:\temp\file.txt", the encrypted file should be:
"C:\temp\file_encrypted.txt".
2. The key should be written to a file named "key.txt" in the same location of the encrypted file.
3. The user should be notified with a message that notes the location of the encrypted file and the key file.
4. In case of Decryption:
a. Inputs the path to the encrypted source file and the key file from the user.
b. Uses the inputted key to decrypt the file.

c. The result should be written to a new file, and the name should be in the next format:

[original-file-name]_**decrypted**.[original file extension].

For example, if the source file name is "C:\temp\file.txt", the encrypted file should be: "C:\temp\file_ decrypted.txt".

1. The user should be notified with a message that notes the location of the decrypted file.

***Important Notes:* (Important to read before implementation)**
Define classes that are responsible for the encryption algorithm and its results.

Use methods to get rid of duplicate code.

For example: Write a class called EncryptionAlgorithm that performs the inner logic if the encryption and a different class that performs the file operations.

Of course that every modular will be accepted.

# Chapter 2 - Object Oriented Programing In Java

**Reading Materials**

- Head First – Java (Read all the listed chapters)
- 7 – Better Living In Objectville
- 8 – Serious Polymorphism

**Excercise:**

Now you should enable the usage of several encryption algorithms.

- Create an interface called EncryptionAlgorithm and a implementer class ShiftUpEncryption that performs the encryption algorithm used in the last exercise. Refactor you code accordingly.
- Create a class called FileEncryptor that receives an EncryptionAlgorithm in the constructor of the class. The class should be responsible of encryption and decryption of files. Your program should use this class to encrypt and decrypt it's files.

An example to the method declarations (You don't have to do it exactly this way, this is written only for clarification):

Java

```
1  public void encryptFile(String originalFilePath, String outputFilePath,
2                  String keyPath)
3
4  public void decryptFile(String encryptedFilePath, String outputFilePath,
5                  String keyPath)
```

- Create another algorithm called ShiftMultiplyEncryption, it should do the same thing as ShiftUpEncryption, but instead of adding the value of the key to each character, it should multiply it's value with the value of the character.
- Think how you can write this class without code duplication of ShiftUpEncryption.
- Create an algorithm called DoubleEncryption that uses an existing EncryptionAlgorithm (not necessarily ShiftUpEncryption) and uses it twice to perform stronger encryption (It should use a **different** key each time you apply the encryption).
- Bonus: Create a class called RepeatEncryption that receives in the constructor a number "n" and an encryption algorithm, and performs the operations "n" times (with the same key).
- Bonus: Create a class called XorEncryption the performs the XOR (exclusive or) between the key and the character.
- Finally the program should use double shift up encryption (DoubleEncryption + ShiftUp)
- **Emphasis:** There is no need to make a menu of choosing different methods of encryption. You should check all the different encryption classes in another way.

# Chapter 3 - JUnit, Mocking, Unit Testing

Skip to end of metadata

Go to start of metadata

**Reading Materials**

- JUnit Tutorial – Search it in the Internet.
- Mockito Tutorial – Search it in the Internet.
- TDD Wiki Value
- Unit Testing Wiki Value

**Excercise:**

1. Write unit testing planning for your previously written code and review it with our mentor before starting to implement it.
2. Write all of your tests via the framework of JUnit (We're talking about unit testing) that performs the coverage to all of your specifications in the previous section.

a. Use Maven to add the dependencies to JUnit 4 and Mockito (You should use the most current versions available).
b. Write unit tests to all of the implementations of EncryptionAlgorithm interface.(Make sure that the tests run each time you make a build with maven)
c. Try writing your tests without code duplication, think how you can reuse your code to the different encryption algorithms.

# Chapter 4 - Exceptions In Java

**Reading Material**

- Head First – Java (Read all the listed chapters)
- 10 – Numbers Matter
- 11 – Risky Behavior

**Excercise**

1. Add exception handling to your encryption program.

- Your encryption algorithms should throw *InvalidEncryptionKeyException* in case the key is not in the correct format.
- Show the user an appropriate error message in that case.
- Your program should throw another kind of exception in case the paths that were passed are not valid.

1.

a. Add a method to EncryptionAlgorithm that is called getKeyStrength that describes that maximal length (number of digits) of the key for the algorithm.
For example if the key is between 0-256 then the value of the algorithm should be 3.

- Implement the property for each and one of you algorithms.
- Create a class that implements "Comparator<T>" and compares between two algorithms by their key strengths.
- Add suitable unit test accordingly (Do you know gow to check if an exceptions is throwr via JUnit?)

# Chapter 5 - Generics & Listeners

**Reading Material:**

1. Head First – Java
   Chapter 16 – Data Structures
2. Design Patterns Java Workbook
   Chapter 9 – Observer
3. Effective Java, 2nd Edition
   Chapter 3 – Methods Common to All Objects (items 8, 9)
   Chapter 5 – Generics (item 25)
4. עקרונות כתיבת לוגים

**Exercise:**

Add the following events to FileEncryptor (choose correct parameter types of each event):

- EncryptionStarted
- EncryptionEnded
- DecryptionEnded
- DecryptionStarted

Activate them in the appropriate places (the events are supposed to be expressed as methods in the interface, as you read in the Observer Design Pattern – which is the Observer and which is the Observable).

How would you know what is the time? (System class).

- Create the class EncryptionLogger that registers to these events and writes messages like: "the encryption for file X with algorithm Y took Z miliseconds. The encrypted file is located in file F".
- Create EncryptionLogEventArgs class that includes the relevant information for each message.
- Create a class called EncryptionLog4JLogger that writes to Log4J's Logger (you must add the Log4J dependency to your pom file).
- Use both a console appender and a file appender.
- Add some logging (according to the עקרונות כתיבת לוגים document). Make sure you use the correct log level: which log level should the log in the first bullet be? For errornous situations – add error logs (fatal or just error?)
  Debug level logs should be added too. How would you disable the debug level logging in production?
- Implement equals and hashCode for EncryptionLogEventArgs (what is the difference between equals and ==? How can we ensure that both options behave the same?)
- Change the Key definition. In IEncryptionAlgorithm, add a generic parameter that will be the key.
  You will have to change all the implementations of the interface and the automatic tests. (Look again at DoubleEncryption. You will have to create a new class that is a key and has 2 keys).
- Explain what does "covariant" mean?

# Chapter 6 - Multithreading

**Reading Material:**

1. Book: Head First – Java
   Chapter 15: Networking And Threads (p. 489 – 528)
2. *Book:* Effective Java, 2nd Edition
   Chapter 10: Concurrency
3. Book: Oracle Java Tutorial
   Java.util.concurrent locks: Condition, Lock, ReadWriteLock, ReentrantLock, ReentrantReadWriteLock.
4. Java Concurrency Tutorial.
5. *Book:* Effective Java, 2nd Edition
   Chapter 4: Methods Common to All Objects (15 items)

**Exercise**

- Add an option to encrypt an entire directory. The directory may contain several files, and the system should encrypt them all with the same key (and the same algorithm).
  The encrypted files shall sit under a sub-directory named "encrypted". Name files shall not be changed.
  Only files with .txt extension should be encrypted. Do not encrypt any sub directories. The key will be saved inside the directory, as key.txt.
- After a single file encryption has completed – inform the user of the event and its duration. After all files have been encrypted – inform the user about the event and its whole duration.
- Because we are dealing with a large number of files, and some may be heavy, you have decided that you want to use multithreading to try and improve the speed of directory encryption.
- Write an interface named IDirectoryProcessor and implement the following classes:
- SyncDirectoryProcessor – responsible of encrypting a directory "normally", with a single thread.
- AsyncDirectoryProcessor – responsible of encrypting a directory concurrently.
- Think: what changes have to be made in EncryptionLogger? (pay attention to locking and the way it has to be done – synchronized? on what?)
- Implement the reverse functionality – decryption of a whole directory, similarly.
- Compare the running time of each method. Apply each encryption method on a directory with some big files (each file should be ~3MB). Selection of the method should be done in the main method.
- Which method is faster? Why?

Important notes:

- In the final implementation you should perform encryption and decryption of a directory asynchronously (AsyncDirectoryProcessor).
- Pay attention to how are the threads being run? Thread? Executor?
- For the sake of testing you probably should write something that generates large files. You can write a little Java program for this task.

# Chapter 7 - JAXB and GSON

XML, XSD, (JAXB, SAX, DOM, STAX)

**Reading Material:**

1. Oracle Java Tutorial – JAXB – read all.

**Exercise**

We want to be able to read program parameters from an XML/JSON file instead of getting it from the user (through the console).

And we want to print the run results to an XML/JSON file.

- We shall implement this functionality in 2 different ways: JAXB, JSON
- 3 different classes should be created for the 3 methods.
- The writing will be performed through JAXB/ GSON only.
- An example for an input file is given in the end of this document (this is only an example. You may choose a different format)
- Add a validation option with XSD/JsonSchema of the XML/JSON. You will write the XSD/JsonSchema.

Important Notes

- Read about the xjc command. What can it do?
- The classes should implement interfaces. This should improve flexibility and allow integration in our code.

XML configuration example:

XHTML

```
1  <ProcessSettings>
2      <Algorithm>DoubleShiftUp</Algorithm>
3      <KeyPath>c:\temp\encryptor</KeyPath>
4      <SourceDirectory>c:\temp\encryptor\source</SourceDirectory>
5      <SourceFileName>c:\temp\encryptor\file.txt</SourceFileName>
6  </ProcessSettings>
```

# Chapter 8 - Reflection & Dependency Injection

**Reading Material:**

1. Jenkov Tutorial – Reflection API
2. SOLID article
3. Martin Fowler – Dependency Injection
4. Design Patterns Java Workbook
   Chapter 16: Factory
5. Guice Tutorial (from the internet)

**Exercise:**
In the previous exercises we used Constructor Injection to pass the algorithm implementation of choice.

Replace this method of DI with Google Guice's @Inject annotation.

Everything should be defined in Guice's AbstractModule (you should inherit from it).

Update the automatic tests to work with the injection configuration using Guice (it is possible to implement this using an inner class in the tests).

The key creation method should be changed to the following method (instead of a random key):

- Read the method names of all the methods in the implementation class of the algorithm (e.g. DoubleEncryption) using reflection.
- Write a hash function that transforms the list of strings (method names) to the key.