# 6. homework assignment; JAVA, Part 1, Academic year 2016/2017; FER

Napravite prazan Maven projekt, kao u 1. zadaći: u Eclipsovom workspace direktoriju napravite direktorij `hw06-0000000000` (zamijenite nule Vašim JMBAG-om) te u njemu oformite Mavenov projekt `hr.fer.zemris.java.jmbag0000000000:hw06-0000000000` (zamijenite nule Vašim JMBAG-om) i dodajte ovisnost prema `junit:junit:4.12`. Importajte projekt u Eclipse. Sada možete nastaviti s rješavanjem zadataka.

Pročitajte posljednju stranicu upute. Jeste? OK; ova zadaća sastoji se od dva dijela.

## *Problem 1.*

You will write a program `Crypto` that will allow the user to encrypt/decrypt given file using the AES crypto-algorithm and the 128-bit encryption key or calculate and check the SHA-256 file digest. Since this kind of cryptography works with binary data, use octet-stream Java based API for reading and writing of files. What needs to be programmed is illustrated by the following use cases (I have skipped *classpath* parameter; set it to appropriate value). You will find in repository additional file which is used in this example: `hw06test.bin`. Download this file and place it in your projects current directory. Program outputs are shown red, user input is shown in blue.

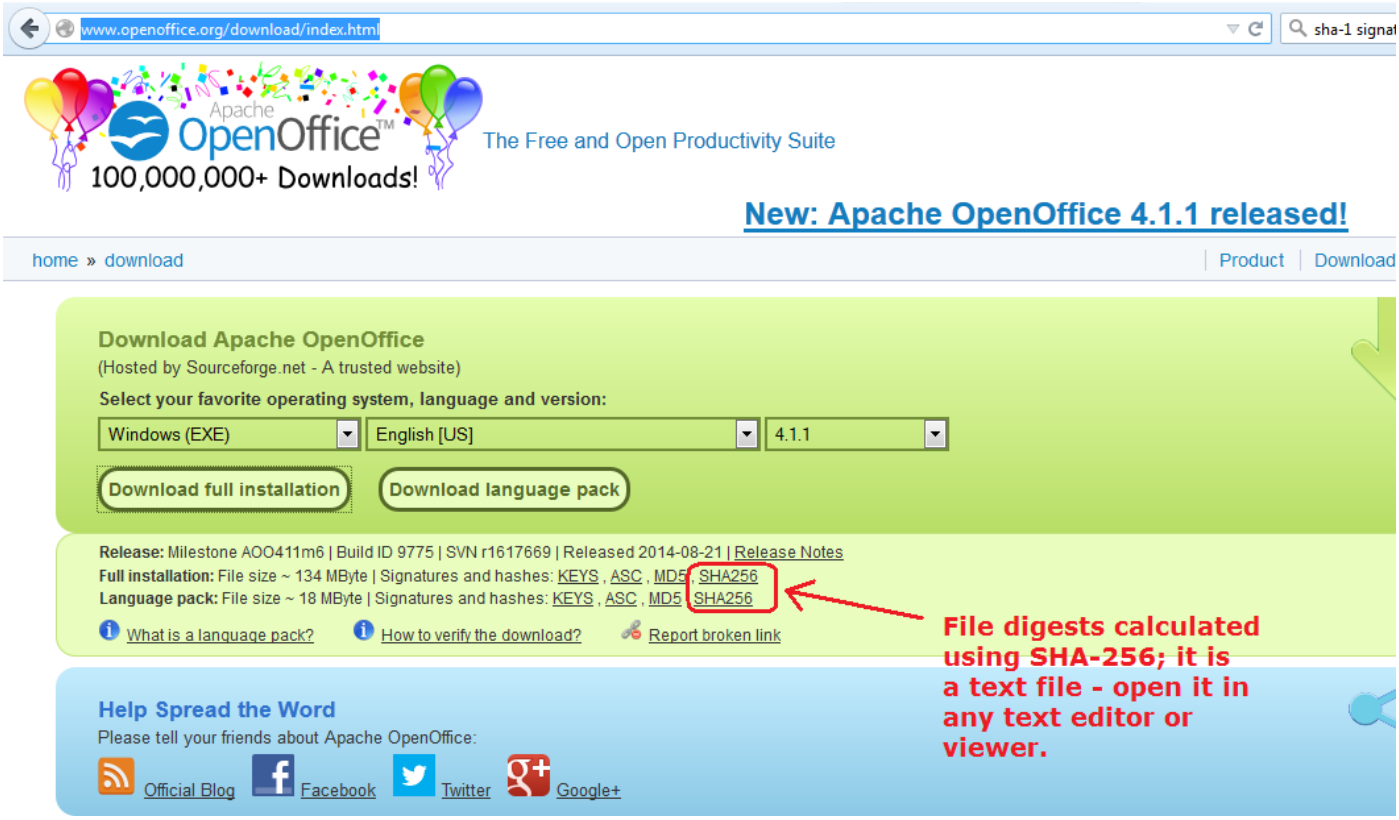
```
java hr.fer.zemris.java.hw06.crypto.Crypto checksha hw06test.bin
Please provide expected sha-256 digest for hw06part2.pdf:
> 2e7b3a91235ad72cb7e7f6a721f077faacfeafdea8f3785627a5245bea112598
Digesting completed. Digest of hw06test.bin matches expected digest.

java hr.fer.zemris.java.hw06.crypto.Crypto checksha hw06test.bin
Please provide expected sha-256 digest for hw06test.bin:
> d03d4424461e22a458c6c716395f07dd9cea2180a996e78349985eda78e8b800
Digesting completed. Digest of hw06test.bin does not match the expected digest. Digest
was: 2e7b3a91235ad72cb7e7f6a721f077faacfeafdea8f3785627a5245bea112598

java hr.fer.zemris.java.hw06.crypto.Crypto encrypt hw06.pdf hw06.crypted.pdf
Please provide password as hex-encoded text (16 bytes, i.e. 32 hex-digits):
> e52217e3ee213ef1ffdee3a192e2ac7e
Please provide initialization vector as hex-encoded text (32 hex-digits):
> 000102030405060708090a0b0c0d0e0f
Encryption completed. Generated file hw06.crypted.pdf based on file hw06.pdf.

java hr.fer.zemris.java.hw06.crypto.Crypto decrypt hw06.crypted.pdf hw06orig.pdf
Please provide password as hex-encoded text (16 bytes, i.e. 32 hex-digits):
> e52217e3ee213ef1ffdee3a192e2ac7e
Please provide initialization vector as hex-encoded text (32 hex-digits):
> 000102030405060708090a0b0c0d0e0f
Decryption completed. Generated file hw06orig.pdf based on file hw06.crypted.pdf.

java hr.fer.zemris.java.hw06.crypto.Crypto decrypt hw06test.bin hw06test.pdf
Please provide password as hex-encoded text (16 bytes, i.e. 32 hex-digits):
> e52217e3ee213ef1ffdee3a192e2ac7e
Please provide initialization vector as hex-encoded text (32 hex-digits):
> 000102030405060708090a0b0c0d0e0f
Decryption completed. Generated file hw06test.pdf based on file hw06test.bin.
```

First two examples test your implementation of digest calculation. Third and fourth example test is your implementation of file encryption and decryption compatible with itself. The fifth example tests is your decryption procedure compatible with the encryption procedure which was done by me. If this last step works, you will be able to open `hw06test.pdf` in PDF viewer and read its content.

Lets just briefly explain some of the concepts from this problem.

*Message digest* is a fixed-size binary digest which is calculated from arbitrary long data. The idea is simple. You have some original data (lets denote it D); this can be a file on disk. Then you calculate a digest for this data (lets denote it S); for example, if S is calculated with SHA-256 algorithm, the digest will always be 256-bits long, no matter how long is the original file you digested. Generally speaking, the original data can not be reconstructed from the digest and this is not what the digests are used for. Digests are used to verify if the data you have received (for example, when downloading the data from the Internet) arrived unchanged. You will verify this by calculating the digest on the file you have downloaded and then you will compare the calculated digest with the digest which is published on the web site from which you have started the download. If something has changed during the download, there is extremely high probability that the calculated digest will be different from the one published on the web site. You can see this on many of web-pages which offer file download. Visit, for example, Open Office download page:

http://www.openoffice.org/download/index.html




*Note:* Digests will be integral part of *digital signature* – a mechanism which is today broadly used online as a replacement for persons physical signature. At FER you will learn more on this if you enroll the course Advanced operating systems (*Computing Master programme*, profile *Computer Science*).

*Encryption* is the conversion of data into a form, called a *ciphertext*, that can not be easily understood by unauthorized people. *Decryption* is the reverse process: it is a transformation of ciphertext back into its original form. There are two families of cryptography: *symmetric* in which both encryption and decryption

use the same key (i.e. “password”), and *asymmetric* in which a pair of keys is used (public key and private key which are mutually inverse: what is encrypted with one can only be decrypted with other). Since the decryption of encrypted data must be possible, there can be no loss of data (as is the case with digests). Encrypted data will always be as big (or even bigger) as were the original data. In this homework we will use a symmetric crypto-algorithm AES which can work with three different key sizes: 128 bit, 192 bit and 256 bit. Since AES is *block cipher*, it always consumes 128 bit of data at a time (or adds padding if no more data is available) and produces 128 bits of encrypted text. Therefore, the length (in bytes) of encrypted file file will always be divisible by 16.

In this homework you are not expected to implement these algorithms. You only have to learn how to use them in your programs. Java already offers appropriate implementations. Please consult the following references:

http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#MessageDigest
http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#Cipher
http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#MDEx
http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#SimpleEncrEx

Encryption keys and initialization vectors are byte-arrays each having 16 bytes. In the above example it is expected from the user to provide these as hex-encoded texts.

Implement these methods. To obtain properly initialized `Cipher` object, use following code snippet:

```
String keyText = … what user provided for password …
String ivText = … what user provided for initialization vector …
SecretKeySpec keySpec = new SecretKeySpec(Util.hextobyte(keyText), "AES");
AlgorithmParameterSpec paramSpec = new IvParameterSpec(hextobyte(ivText));
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, paramSpec);
```

Create a class `Util` with two public static methods: `hextobyte(keyText)` and `bytetohex(bytearray)`. Method `hextobyte(keyText)` should take hex-encoded String and return appropriate `byte[]`. If string is not valid (odd-sized, has invalid characters, …) throw an `IllegalArgumentException`. For zero-length string, method must return zero-length byte array. Method must support both uppercase letters and lowercase letters.

For example: `hextobyte("01aE22")` should return `byte[] {1, -82, 34}`.

Method `bytetohex(bytearray)` takes a byte array and creates its hex-encoding: for each byte of given array, two characters are returned in string, in big-endian notation. For zero-length array an empty string must be returned. Method should use lowercase letters for creating encoding.

For example: `bytetohex(new byte[] {1, -82, 34})` should return `"01ae22"`.

You are expected to write your own implementation of these two methods (using other classes for this is not allowed). Write unit tests for this methods to ensure they work correctly.

Please note, **you are not allowed** to use `CipherInputStream` or `CipherOutputStream` (or any of its subclasses); you are required to implement encryption/decryption directly using `Cipher` object and a series of `update/update/update/...` completed by `doFinal()`. Also, you are not allowed to read a complete file into memory, then encrypt/decrypt it and then write the result back to disk since the input file can be huge. You are only allowed to read a reasonable amount of file into memory at each single time (for example, 4k) – use byte streams for this. The same goes for constructing the resulting file. For reading file you must use an instance of `FileInputStream` and for writing an instance of `FileOutputStream`. Ensure that they are both buffered – use appropriate method for constructing buffered file streams directly from `Files` class.

Be aware that algorithms we use here for encryption and decryption are block-based. They must get a block of bytes in order to create new encrypted/decrypted block of bytes. This is the reason for existence of methods `update` and `doFinal`. When you pass some bytes by calling `update`, algorithm processes as many blocks as possible and return processed blocks while retaining still-to-process bytes in internal buffer. When you call `update` again and deliver new bunch of bytes, processing continues and you get new processed blocks as result. You repeat this in loop, reading data from input stream, and writing processed data to output stream. Once the input stream is drained, you must complete processing by calling `doFinal`. Using this call, you signal to the encryption/decryption procedure that there will be no more data, and that in order to complete processing, it should itself add padding in needed in order to create last block of data and then process it, returning to you the processed data.

Taking this into account, do not be surprised if you find out that the encrypted file is a bit larger then the original file: its size will be multiple of algorithm block size. When performing decryption, the algorithm will be aware that the padding was added, and you will obtain only the data you sent in when encrypting: reconstructed file will have the same size as the original file.

## *Problem 2.*

Download the file `hw06part2.bin` from Ferko repository and save it in you current directory. Now run your program:

```
java hr.fer.zemris.java.hw06.crypto.Crypto checksha hw06part2.bin
Please provide expected sha-256 digest for hw06part2.bin:
> 603ce08075a10ea3f781301bfafc01e3e9c9487ba33790d4afa7fd15dffd2b94
Digesting completed. Digest of hw06part2.bin matches expected digest.
```

If you obtain different result, there is something wrong; either the file `hw06part2.bin` is corrupted (redownload it again) or you have bug in your program (fix it). When you do obtain result as expected, run following command:

```
java hr.fer.zemris.java.hw06.crypto.Crypto decrypt hw06part2.bin hw06part2.pdf
Please provide password as hex-encoded text (16 bytes, i.e. 32 hex-digits):
> e52217e3ee213ef1ffdee3a192e2ac7e
Please provide initialization vector as hex-encoded text (32 hex-digits):
> 000102030405060708090a0b0c0d0e0f
Decryption completed. Generated file hw06part2.pdf based on file hw06part2.bin.
```

Open the file you just generated, read it and proceed as instructed by the text in that file.

**Please note.** You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries for this homework (whether it is yours old code or someones else). From now on, you can use Java Collection Framework and other parts of Java covered by lectures; if unsure – e-mail me. Document your code!

**If you need any help, I have reserved a slot for consultations Monday at 1 PM, Tuesday at 2 PM, and Wednesday at 1 PM. Feel free to drop by my office.**

All source files must be written using UTF-8 encoding. All classes, methods and fields (public, private or otherwise) must have appropriate javadoc.

You are expected to write tests for Util class methods.
You are encouraged to write tests for other problems.

Once you complete the problems 1 and 2, you will discover what is left, in order to complete the entire homework.

When your **complete** homework is done, pack it in zip archive with name `hw06-0000000000.zip` (replace zeros with your JMBAG). Upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is April 13th 2017. at 07:00 AM.

# 6. homework assignment; JAVA, Part 2, Academic year 2016/2017; FER

## *Problem 3.*

Write a command-line program `MyShell` and put it in package `hr.fer.zemris.java.hw06.shell`. When started, your program should write a greeting message to user (`Welcome to MyShell v 1.0`), write a prompt symbol and wait for the user to enter a command. The command can span across multiple lines. However, each line that is not the last line of command must end with a special symbol that is used to inform the shell that more lines as expected. We will refer to these symbols as `PROMPTSYMBOL` and `MORELINESSYMBOL`. For each line that is part of multi-line command (except for the first one) a shell must write `MULTILINESYMBOL` at the beginning followed by a single whitespace. Your shell must provide a command symbol that can be used to change these symbols. See example (*classpath* is omitted; set as appropriate):

```
C:\Users> java hr.fer.zemris.java.hw06.shell.MyShell
Welcome to MyShell v 1.0
> symbol PROMPT
Symbol for PROMPT is '>'
> symbol PROMPT #
Symbol for PROMPT changed from '>' to '#'
# symbol \
| MORELINES \
| !
Symbol for MORELINES changed from '\' to '!'
# symbol !
| MORELINES
Symbol for MORELINES is '!'
# symbol MULTILINE
Symbol for MULTILINE is '|'
# exit
C:\Users>
```

In order to make your shell usable, you must provide following built-in commands: `charsets`, `cat`, `ls`, `tree`, `copy`, `mkdir`, `hexdump`.

Command `charsets` takes no arguments and lists names of supported charsets for your Java platform (see `Charset.availableCharsets()`). A single charset name is written per line.

Command `cat` takes one or two arguments. The first argument is path to some file and is mandatory. The second argument is charset name that should be used to interpret chars from bytes. If not provided, a default platform charset should be used (see `java.nio.charset.Charset` class for details). This command opens given file and writes its content to console.

Command `ls` takes a single argument – directory – and writes a directory listing (not recursive). Output should be formatted as in following example.

```
-rw-      53412 2009-03-15 12:59:31 azuriraj.ZIP
drwx       4096 2011-06-08 12:59:31 b
drwx       4096 2011-09-19 12:59:31 backup
-rw-   17345597 2009-02-18 12:59:31 backup-ferko-20090218.tgz
drwx       4096 2008-11-09 12:59:31 beskonacno
drwx       4096 2010-10-29 12:59:31 bin
-rwx        282 2011-02-10 12:59:31 burza.sh
```

```
-rwx        281 2011-02-10 12:59:31 burza.sh~
-rwx       1316 2009-09-10 12:59:31 burza_stat.sh
drwx       4096 2011-09-02 12:59:31 ca
drwx       4096 2008-09-02 12:59:31 CA
-rw-          0 2008-09-02 12:59:31 ca.key
```

The output consists of 4 columns. First column indicates if current object is directory (`d`), readable (`r`), writable (`w`) and executable (`x`). Second column contains object size in bytes that is right aligned and occupies 10 characters. Follows file creation date/time and finally file name.

To obtain file attributes (such as creation date/time), see the following snippet.

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
Path path = Paths.get("d:/tmp/javaPrimjeri/readme.txt");
BasicFileAttributeView faView = Files.getFileAttributeView(
      path, BasicFileAttributeView.class, LinkOption.NOFOLLOW_LINKS
);
BasicFileAttributes attributes = faView.readAttributes();
FileTime fileTime = attributes.creationTime();
String formattedDateTime = sdf.format(new Date(fileTime.toMillis()));
System.out.println(formattedDateTime);
```

The `tree` command expects a single argument: directory name and prints a tree (each directory level shifts output two charatcers to the right).

The `copy` command expects two arguments: source file name and destination file name (i.e. paths and names). Is destination file exists, you should ask user is it allowed to overwrite it. Your `copy` command must work only with files (no directories). If the second argument is directory, you should assume that user wants to copy the original file into that directory using the original file name. You must implement copying yourself: you are not allowed to simply call copy methods from `Files` class.

The `mkdir` command takes a single argument: directory name, and creates the appropriate directory structure.

Finally, the `hexdump` command expects a single argument: file name, and produces hex-output as illustrated below. On the right side of the image only a standard subset of characters is shown; for all other characters a '.' is printed instead (i.e. replace all bytes whose value is less than 32 or greater than 127 with '.').

```
00000000: 31 2E 20 4F 62 6A 65 63|74 53 74 61 63 6B 20 69 | 1. ObjectStack i
00000010: 6D 70 6C 65 6D 65 6E 74|61 63 69 6A 61 0D 0A 32 | mplementacija..2
00000020: 2E 20 4D 6F 64 65 6C 2D|4C 69 73 74 65 6E 65 72 | . Model-Listener
00000030: 20 69 6D 70 6C 65 6D 65|6E 74 61 63 69 6A 61 0D |  implementacija.
00000040: 0A                     |                       | .
```

If user provides invalid or wrong argument for any of commands (i.e. user provides directory name for `hexdump` command), appropriate message should be written and your shell should be prepared to accept a new command from user. Shell terminates when user gives `exit` command.

You should allow user to use quotes in paths, so that a paths containing spaces can be given. For example, following commands should be accepted.

```
copy /home/john/info.txt /home/john/backupFolder
copy "C:/Program Files/Program1/info.txt" C:/tmp/informacije.txt
```

How should you organize your code? Start by defining an interface `Environment`:

```
String readLine() throws ShellIOException;
void write(String text) throws ShellIOException;
void writeln(String text) throws ShellIOException;
SortedMap<String, ShellCommand> commands();
Character getMultilineSymbol();
void setMultilineSymbol(Character symbol);
Character getPromptSymbol();
void setPromptSymbol(Character symbol);
Character getMorelinesSymbol();
void setMorelinesSymbol(Character symbol);
```

This is an abstraction which will be passed to each defined command. The each implemented command communicates with user (reads user input and writes response) **only through this interface**. Method commands() must return unmodifiable map, so that the client can not delete commands by clearing the map; please see java.util.Collections and method unmodifiableSortedMap, which builds a proxy object (a wrapper).

Methods Environment.readLine() / Environment.write / Environment.writeln are used for reading from and writing to user (console). Define a new exception: ShellIOException (subclass of RuntimeException). If reading or writing fails, throw this exception.

Define an interface `ShellCommand` that has following methods:

```
ShellStatus executeCommand(Environment env, String arguments);
String getCommandName();
List<String> getCommandDescription();
```

The second argument of method `executeCommand` is a single string which represents everything that user entered AFTER the command name. It is expected that in case of multiline input, the shell has already concatenated all lines into a single line and removed `MORELINES` symbol from line endings (before concatenation). This way, the command will always get a single line with arguments. Method `getCommandName()` returns the name of the command while `getCommandDescription()` returns a description (usage instructions); since the description can span more than one line, a read-only `List` must be used (either create read-only List decorator or check class Collections).

Implement the `help` command. If started with no arguments, it must list names of all supported commands. If started with single argument, it must print name and the description of selected command (or print appropriate error message if no such command exists).

`ShellStatus` should be enum `{CONTINUE, TERMINATE}`.

Implement each shell command as a class that implements `ShellCommand` interface. During shell startup, build a map of supported commands:

```
SortedMap<String, ShellCommand> commands = …;
commands.put("exit", new ExitShellCommand());
commands.put("ls", new LsShellCommand());
...
```

Then implement the shell as illustrated in the following pseudocode:

```
build environment
repeat {
```

```
  l = readLineOrLines
  String commandName = extract from l
  String arguments = extract from l
  command = commands.get(commandName)
  status = command.executeCommand(environment, arguments)
} until status!=TERMINATE
```

*Note:* the responsibility for argument splitting is on the commands themselves. Theoretically, each command can define a different way how the splitting of its arguments must be done. This, however, does not mean that you must duplicate this functionality in all commands. Commands can use shared utility classes if they use same rules for argument splitting. In command where a file-path is expected, you must support quotes to allow paths with spaces (such as `"Documents and Settings"`). In order to do so, if argument starts with quotation, you must support during parsing `\"` as escape sequence representing `"` as regular character (and not string end). Additionally, a sequence `\\` should be treated as single `\`. Every other situation in which after `\` follows anything but `"` and `\` should be literally copied as two characters, so that you can write `"C:\Documents and Settings\Users\javko"`. Also, the simbol for `MORELINES` has only a special meaning if it is the last character in line. Please note, the described escaping mechanism is supported ONLY inside strings started with double quote. After the ending double-quote, either no more characters must be present or at least one space character must be present: strings like `"C:\fi le".txt` are invalid and argument parser should report an error.

Place command implementations into subpackage `hr.fer.zemris.java.hw06.shell.commands`.

Implementation details

Your commands should not throw exceptions that would propagate back to the Shell. For example, if a command is `cat`, and file could not be opened, the command itself must write an appropriate message to the user and return to shell with status `CONTINUE`. If there is an exception when reading from or writing to user using `Environment.readLine()` / `Environment.write()` / `Environment.writeln()`, you can propagate that exceptions to shell. When shell catches `ShellIOException`, it must terminate, since no communication is possible with the user.

**Important notes**

You must create a single ZIP archive containing all projects which you have created as part of this homework (each in its own folder), and than upload this single ZIP. ZIP arhive must have name `HW06-`*yourJMBAG*`.zip`.

**Please note.** You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries which is not part of Java standard edition (Java SE) unless explicitly allowed or provided by me. You can use Java Collection Framework classes and its derivatives. Document your code!

Upload final ZIP archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. The deadline is defined in the first part of this homework assignment.