# Object-Oriented Programming – Sample Exam

## Problem 1 – Document System

A **document system** holds a list of **documents**. Documents can be **binary** or **text** and have **name** (mandatory) and can have **content**. Documents are of two **types**: **text** and **binary**. Text documents can have **charset** (e.g. utf-8 or windows-1251). Binary documents can have **size** (in bytes). Binary documents can be of type **PDF**, **Word**, **Excel**, **Audio** or **Video**. PDF documents can hold the **number of pages** they consist of. Word documents can hold the **number of characters** they consist of. Excel documents can hold the **number of rows** and the **number of columns** in the table they hold. Word and Excel documents are both **office documents**. Office document can have **version** (e.g. “2007” or “Office97”). A special kind of binary documents are the **multimedia documents**. All multimedia documents can have **length** (in seconds). **Audio** documents and **video** documents are both multimedia documents. Audio documents can have **sample rate** (in Hz). Video documents can have **frame rate** (in fps). PDF, Word and Excel documents are **encryptable** (can be encrypted and decrypted). Word and text documents are **editable** (their content could be changed). All document characteristics except their name are **non-mandatory**.

### Design the Class Hierarchy

Your **first task** is to **design an object-oriented class hierarchy** to model the document system and the documents it can hold using the best practices for object-oriented design (OOD) and object-oriented programming (OOP). Additionally you are given few C# **interfaces** that you should obligatory use:

|  |
| --- |
| using System.Collections.Generic;  public interface **IDocument**  {  string Name { get; }  string Content { get; }  void LoadProperty(string key, string value);  void SaveAllProperties(IList<KeyValuePair<string, object>> output);  string ToString();  }  public interface **IEditable**  {  void ChangeContent(string newContent);  }  public interface **IEncryptable**  {  bool IsEncrypted { get; }  void Encrypt();  void Decrypt();  } |

All your documents should implement IDocument. It specifies that documents have immutable Name and Content, can load their properties from key-value pairs through the LoadProperty(key, value) method and save their properties in a list of key-value pairs through the SaveAllProperties(…) method as well as be printed on the console through the ToString() method.

All editable documents should implement the IEditable interface. All changes of the document content should pass through this interface (direct content changes are not allowed).

All encryptable documents should implement the IEncryptable interface. You do not need to implement encryption algorithm (like AES and Blowfish), just to keep whether a document is encrypted or not in its internal state. You are allowed to encrypt / decrypt a document only though this interface.

### Write a Program to Execute Commands

Your **second task** is to write a program that executes a sequence of up to 1000 commands. Each command is given in the following format:

|  |
| --- |
| Command[key1=value1;key2=value2;…] |

Commands consist of command **name** and **attributes** given in square brackets **[** **]**. Each attribute is given in the form **key=value**. **Keys** consist of small English letters. **Values** consist single line English text and cannot contain the following characters: [, ], =, ; and \n. The sequence of commands ends with an empty line.

The valid commands that your program should be able to execute are the following:

* **AddTextDocument[name=…;charset=…;content=…]** – adds a text document to the document system. The **name** is obligatory, but all other attributes are optional. The order of the attributes can be arbitrary. All attributes will be valid for the type of the document we create. As a result the command prints on the console “**Document added: <name>**” is case of success or “**Document has no name**” in case of missing document name. Multiple documents having the same name are allowed to be added.
* **AddPDFDocument[name=…;pages=…;size=…;content=…]** – works like **AddTextDocument**.
* **AddWordDocument[chars=…;name=…;version=…;size=…;content=…]** – works like **AddTextDocument**.
* **AddExcelDocument[rows=…;cols=…;version=…;size=…;name=…;content=…]** – works like **AddTextDocument**.
* **AddAudioDocument[name=…;content=…;samplerate=…;length=…;size=…]** – works like **AddTextDocument**.
* **AddVideoDocument[name=…;content=…;framerate=…;length=…;size=…]** – works like **AddTextDocument**.
* **EncryptDocument[name]** – changes the state of all documents matching the specified name to “encrypted”. Documents that are already encrypted remain in “encrypted” state. For each document matching the specified name, the command prints as a result “**Document encrypted: <name>**” on the console or prints “**Document does not support encryption: <name>**” if the document is not encryptable. In case of no document is matching the specified name, the message “**Document not found: <name>**”.
* **DecryptDocument[name]** – works similarly like **EncryptDocument**, but changes the state of all matched documents to “not encrypted” and prints as result one of the following messages: “**Document decrypted: <name>**”, “**Document does not support decryption: <name>**” or “**Document not found: <name>**”.
* **EncryptAllDocuments** – changes the state of all documents that support encryption to “encrypted”. As result, if at least one document supports encryption, prints on the console “**All documents encrypted**”, otherwise prints “**No encryptable documents found**”.
* **ListDocuments[]** – prints on the console all the documents in the document system in their order of their addition. Each document should be printed alone on a single line in the following form:

|  |
| --- |
| XXXDocument[key1=value1;key2=value2;…] |

The XXXDocument is the type of the document, e.g. PDFDocument, VideoDocument, etc. The keys should be **ordered** alphabetically. Keys with no value should be skipped. In there are no documents, the command prints “**No documents found**”. Encrypted documents are shown differently, as follows:

|  |
| --- |
| XXXDocument[encrypted] |

* **ChangeContent[name;new\_content]** – changes the content of all editable documents matching the specified name with the specified new content. For each document matching the specified name, the command prints as a result “**Document content changed: <name>**” on the console or prints “**Document is not editable: <name>**” if the document is not editable. In case of no document is matching the specified name, the message “**Document not found: <name>**”.

The commands are guaranteed to be **valid**. Only the described above commands will be given as input. The command format will be as described above. The command parameters will also be in the described format. All attributes will be valid for their corresponding command. The commands will be no more than **1000**. Each command will be less than **500 characters** long. To simplify your work you are given a command parser that provides a skeleton for your solution (see the file DocumentSystem-Skeleton.rar).

|  |  |  |  |
| --- | --- | --- | --- |
| **Sample input:**   |  | | --- | | **AddTextDocument[name=example.txt;charset=utf-8;content=Telerik Academy Exam]**  **AddTextDocument[name=readme.txt]**  **AddTextDocument[]**  **EncryptAllDocuments[]**  **AddPDFDocument[content=6A7E889CF3A8D2;name=academy.pdf;pages=2;size=38217]**  **AddWordDocument[name=exam.docx;chars=12218;version=2012;size=36881]**  **AddWordDocument[name=exam.docx]**  **AddExcelDocument[name=academy.xls;rows=12;cols=3;size=9430;version=97]**  **AddAudioDocument[size=9834733;name=ring.mp3;samplerate=44100;length=368800]**  **AddVideoDocument[name=demo.mpg;framerate=24;length=87312;size=87245212]**  **EncryptDocument[academy.pdf]**  **EncryptDocument[ring.mp3]**  **EncryptDocument[exam.docx]**  **EncryptDocument[invalid.doc]**  **ChangeContent[example.txt;new content]**  **ChangeContent[demo.mpg;new content]**  **ChangeContent[invalid.doc;new content]**  **EncryptAllDocuments[]**  **DecryptDocument[academy.pdf]**  **DecryptDocument[exam.docx]**  **DecryptDocument[ring.mp3]**  **DecryptDocument[invalid.doc]**  **ListDocuments[]**  ***(empty line)*** | | **Sample output:**   |  | | --- | | **Document added: example.txt**  **Document added: readme.txt**  **Document has no name**  **No encryptable documents found**  **Document added: academy.pdf**  **Document added: exam.docx**  **Document added: exam.docx**  **Document added: academy.xls**  **Document added: ring.mp3**  **Document added: demo.mpg**  **Document encrypted: academy.pdf**  **Document does not support encryption: ring.mp3**  **Document encrypted: exam.docx**  **Document encrypted: exam.docx**  **Document not found: invalid.doc**  **Document content changed: example.txt**  **Document is not editable: demo.mpg**  **Document not found: invalid.doc**  **All documents encrypted**  **Document decrypted: academy.pdf**  **Document decrypted: exam.docx**  **Document decrypted: exam.docx**  **Document does not support decryption: ring.mp3**  **Document not found: invalid.doc**  **TextDocument[charset=utf-8;content=new content;name=example.txt]**  **TextDocument[name=readme.txt]**  **PDFDocument[content=6A7E889CF3A8D2;name=academy.pdf;pages=2;size=38217]**  **WordDocument[chars=12218;name=exam.docx;size=36881;version=2012]**  **WordDocument[name=exam.docx]**  **ExcelDocument[encrypted]**  **AudioDocument[length=368800;name=ring.mp3;samplerate=44100;size=9834733]**  **VideoDocument[framerate=24;length=87312;name=demo.mpg;size=87245212]** | |