Kristina Myftaraga

200069406

Prof. Vladimir Georgiev

March 21, 2020

Fundamental Data Structures

Course Project Phase 2: Program Design

Editors

Assignment: *“Submit a document describing the abstract data types for your classes,*

*hierarchy of classes, description and evaluation of the algorithms that you are going to*

*use.”*

**Hierarchy of classes:**

1. For the ADT and classes - describe the data, important axioms and parent class(es).

**class Editor {** -> This is the parent class.

**public**: -> public type so the children can have access

Editor();

// get and set functions:

**void** setDesiredPrice() {}

**void** setName() {}

**void** setVersion() {}

**void** setFormat() {}

**void** setOsystem() {}

**void** setCompany(Company\* object) {

company = object;

}

**void** setID() {}

Company\* getCompany() { return company; }

string getEversion() { return eVersion; }

string getEformat() { return eFormat; }

string getoSystem() { return eOsystem; }

string getEname() { return eName; }

string getID() { return ID; }

// polymorphic functions

virtual void Description() {

cout << "Name: " << eName;

cout << "\nVersion: " << eVersion;

cout << "\nFormat: " << eFormat << endl;

cout << " Operating system : " << eOsystem << endl;

}

virtual double getTotalPrice() = 0;

};

**----------------------------------------------------------------------------------**

// Level 2:

**class** PhotoEditor : **public** Editor {-> PhotoEditor class, child of Editor

**public**:

PhotoEditor() {

setOriginalSaved();

setColorMode();

setLayers();

}

//constructor to read from the file

PhotoEditor(string NAME, string VERSION, string id, string FORMAT, **double** DW, string OSYSTEM, **bool** \_OriginalSaved, string \_ColorMode, **int** layers):

Editor(NAME, VERSION, FORMAT, id, DW, OSYSTEM),

OriginalSaved(\_OriginalSaved),

ColorMode(\_ColorMode),

Layers (layers){}

**void** setOriginalSaved(){}

**void** setColorMode() {}

**void** setLayers() {}

**bool** getOriginalSaved() { **return** OriginalSaved; }

**string** getColorMode() { **return** ColorMode; }

**int** getLayers() { **return** Layers; }

**void** Description() {

Editor::Description();

cout << " OriginalSaved or Not " << OriginalSaved << endl;

cout << " Select the color mode: " << ColorMode << endl;

cout << " The number of layers " << Layers << endl;}

**private**:

**bool** OriginalSaved;

string ColorMode;

**int** Layers;

};

**-----------------------------------------------------------------------------------------**

**class** VideoEditor : **public** Editor {-> VideoEditor class, child of Editor

**public**:

VideoEditor() {

setAudioMaster();

setMontage();

setOverlay();

}

VideoEditor(string NAME, string VERSION, string id, string FORMAT, **double** DW, string OSYSTEM, **bool** am, string montage, string overlay):

Editor(NAME, VERSION, id, FORMAT, DW, OSYSTEM),

AudioMaster(am),

Montage(montage),

Overlay(overlay){}

**void** setMontage() {}

**void** setOverlay() {}

**void** setAudioMaster(){}

**string** getMontage() { **return** Montage; }

**string** getOverlay() { **return** Overlay; }

**bool** getAudioMaster() { **return** AudioMaster; }

**void** Description() {

Editor::Description();

cout << "montage " << getMontage() << endl;

cout << "overlay " << getOverlay() << endl;

cout << "Do you want the switch to Audio Master? " << getAudioMaster() <<endl;

}

**private**:

string Montage;

string Overlay;

**bool** AudioMaster;

};

**-----------------------------------------------------------------------------------------**

// Level 3:

**class** Simple : **public** PhotoEditor {-> Simple class, child of PhotoEditor

**public**:

Simple() {

setCustomShape();

setExposure();

setFilter();

}

Simple(string NAME, string VERSION, string id, string FORMAT, **double** DW,string OSYSTEM, string FILTER, **bool** OriginalSaved, string ColorMode, **int** layers, string csh, **double** exposure) :PhotoEditor(NAME, VERSION, id, FORMAT, DW, OSYSTEM, OriginalSaved, ColorMode, layers),

CustomShape(csh),

Filter (FILTER),

Exposure (exposure) {}

**void** setCustomShape() {}

**void** setExposure() {}

**void** setFilter() {}

**double** getTotalPrice(){**return** 10;}

string getCustomShape() { **return** CustomShape; }

**double** getExposure() { **return** Exposure; }

string getFilter() { **return** Filter; }

**void** Description() {

PhotoEditor::Description();

cout << " Filters availbale: " << getFilter() << endl;

cout << " Do you want the original saved? " << getOriginalSaved() <<endl;

cout << " Custom Shape of the product: " << getCustomShape() <<endl;

cout << " Exposure level: " << getExposure() << endl;}

**private**:

string CustomShape;

**double** Exposure;

string Filter;

};

**-----------------------------------------------------------------------------------------**

**class** Complex : **public** PhotoEditor {-> Complex class, child of PhotoEditor class

**public**:

Complex() {

setMobileFormat();

setPortraitCorrection();

setPhotoMasking();

setCatalog();

setTraceImage();

setVector();}

Complex(string NAME, string VERSION, string id, string FORMAT, **int** DW, string OSYSTEM, **bool** OriginalSaved, string ColorMode, **int** layers, string mf, **bool** PortraitCorrection, string catalog, string masking, string traceimage, string \_vector):

PhotoEditor(NAME, VERSION, id, FORMAT, DW, OSYSTEM, OriginalSaved, ColorMode, layers),

MobileFormat(mf),

PortraitCorrection(PortraitCorrection),

Catalog (catalog),

PhotoMasking (masking),

TraceImage (traceimage),

Vector (\_vector){}

**void** setMobileFormat() {}

**void** setPhotoMasking() {}

**void** setPortraitCorrection() {}

**void** setCatalog() {}

**void** setTraceImage() {}

**void** setVector() {}

string getMobileFormat() { **return** MobileFormat; }

**bool** getPortraitCorrection() { **return** PortraitCorrection; }

string getCatalog() { **return** Catalog; }

string getPhotoMasking() { **return** PhotoMasking;}

string getTraceImage() { **return** TraceImage; }

string getVector() { **return** Vector; }

**double** getTotalPrice(){}

**void** Description() {PhotoEditor::Description();}

**private**:

string MobileFormat;

**bool** PortraitCorrection;

string Catalog;

string PhotoMasking;

string TraceImage;

string Vector;

};

**-----------------------------------------------------------------------------------------**

**class** SocialMedia : **public** VideoEditor {-> SocialMedia class, child of VideoEditor class

**public**:

SocialMedia() {

setMotionGraphics();

setVisualEffect();

}

SocialMedia(string NAME, string VERSION, string id, string FORMAT, **double** DW, string OSYSTEM, **bool** am, string montage, string overlay, string motiongraphics, string visualeffect):

VideoEditor(NAME, VERSION, id, FORMAT, DW, OSYSTEM, am, montage, overlay),

MotionGraphics (motiongraphics),

VisualEffect (visualeffect)

{}

**void** setMotionGraphics() {}

**void** setVisualEffect() {}

**double** getTotalPrice(){}

**void** Description() {Editor::Description();}

string getMotionGraphics() { **return** MotionGraphics; }

string getVisualEffect() { **return** VisualEffect; }

**private**:

string MotionGraphics;

string VisualEffect;

};

**-----------------------------------------------------------------------------------------**

**class** VirtualReality : **public** VideoEditor {->VirtualReality class, child of VideoEditor class

**public**:

VirtualReality() {

setPlatforms();

setTransitions();

setTimeline();

setStory();}

VirtualReality(string NAME, string VERSION, string id, string FORMAT, **double** DW, string OSYSTEM, **bool** am, string montage, string overlay,string platforms, string transitions, string story, string timeline):

VideoEditor(NAME, VERSION, id, FORMAT, DW, OSYSTEM, am, montage, overlay),

Platforms (platforms),

Transitions (transitions),

timeline (timeline),

story (story){}

**void** setStory() {}

**void** setPlatforms() {}

**void** setTransitions() {}

**void** setTimeline() {}

**double** getTotalPrice(){}

**void** Description() {Editor::Description();}

string getPlatforms() { **return** Platforms; }

string getTransitions() { **return** Transitions; }

string getStory() { **return** story; }

string getTimeline() { **return** timeline; }

**private**:

string Platforms;

string Transitions;

string story;

string timeline;

};

**Important axioms in order the program to work correctly:**

* Since the user input will be read from a text file on the top of it there will be written the following axiom: **Input the data in the order as follows – position name description price. One specific part per line and single space delimited.**
* The price should be calculated as follows: **initial price \* dismantle expanses if any**

Note! The calculation method of the price may be subject of change with the development process of the project

1. *Algorithm description and evaluation should point out what algorithm you will use and for what, as well as its execution time in Big-Oh notation.*

**The sort and search will be implemented through MergeSort and BM pattern finding algorithm.**

* **MergeSort:** I will use merge sort to implement the requirement of the user to have option for sorting in descending and ascending price the Editors.
* **MergeSort** is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves.
* **MergeSort** pseudo code:

**A screenshot of a cell phone

Description automatically generated**

**BM:** I will use BM, because it deals better than KMP in longer alphabets, to implement the requirements of the user to have option for search in the software

**BM:** The idea of bad character heuristic is simple. The character of the text which doesn’t match with the current character of pattern is called the Bad Character. Whenever a character doesn’t match, we slide the pattern in such a way that aligns the bad character with the last occurrence of it in pattern. We preprocess the pattern and store the last occurrence of every possible character in an array of size equal to alphabet size. If the character is not present at all, then it may result in a shift by m (length of pattern). Therefore, the bad character heuristic takes O (n/m) time in the best case.

**BM** I have used BM pattern finding algorithm in previous projects and it works perfectly fine:

A screenshot of a cell phone

Description automatically generated