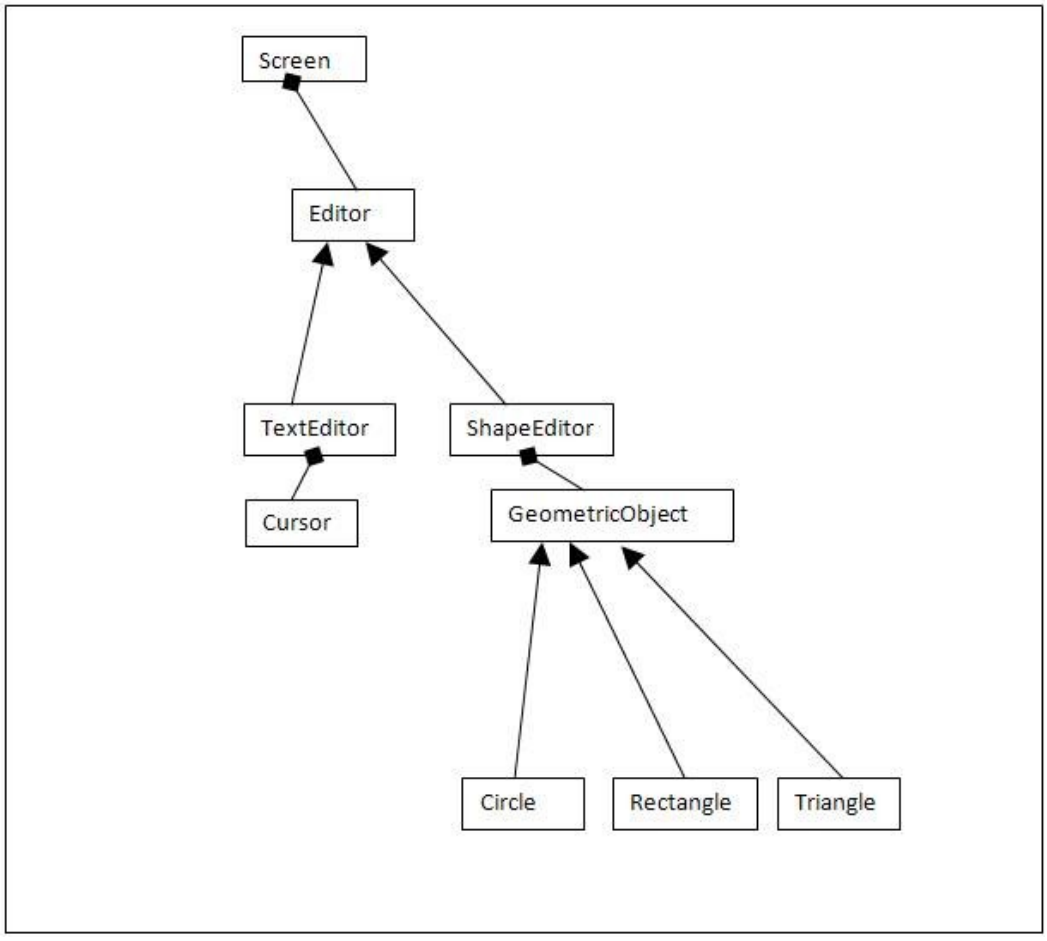
CENG 242

Homework #5

(Due: May 25th Monday 23:55)

In this homework, you are going to implement classes and their methods for a screen framework where you add editors on the screen and make manipulations on them.

`TextEditor` and `ShapeEditor` are the two kinds of editors used in the system which inherit members and functions of `Editor` class. A `Screen` can include any number of editors. A `TextEditor` includes a structure to keep strings in which the inserted text is kept and one `Curser` object. The `Curser` object is used to wandering around the text inserted and selecting any number of characters. The `ShapeEditor` can include any number of `GeometricObjects` which can be of type `Circle`, `Rectangle` or `Triangle`. Below is the figure representing the object relations of the classes.



Event enumerator is used to keep the values in order to handle events for the Editor class.

```
enum Event={COPY, CUT, PASTE, SAVE, CLOSE, DELETE };
```

CLOSE event closes the editor by saving of the features of the objects to its file and it removes the editor from the

environment.

The members of `Direction` enumerator is for pointing out the rotation direction of the `Geometric Object` subclasses.

```
enum Direction {CLOCKWISE, ANTICLOCKWISE};
```

The `Color` enumerator is used to assign colors to the geometric shapes inserted into a `ShapeEditor` instance.

```
enum Color{WHITE,RED,BLUE,GREEN,MAGENTA,CYAN,YELLOW,PINK,BROWN,ORANGE};
```

`Point` and `Error` structors, as in the previos homework are used for defining points in 2D and throwing errors for improper cases respectively.

```
struct Point {

double coordx;

double coordy;

Point (double a, double b) {coordx=a; coordy=b;}

};
```

```
struct Error {

const char *mess;

Error(const char *p) { mess=p;}

};
```

`addEditor` method does not allow to create an editor with the same name with a previosly created one. If such a case occurs, the program should throw an exception saying "Error in Creating the Editor".

Class Definitions:

- A `Screen` class may include any number of editors by adding them with `addEditor` method and giving a different name to each editor. In the case that an editor is intended to be created by the same name with a previously created one, an exception is thrown during the program. Removing of an editor from the screen is done via the name of the editor likewise. A screen can send an event to an editor from the Event enumerator. You can keep the editors list in any structure you want (linked list, vector, array …etc).

```
class Screen {

public:
   Screen();
   void addEditor(Editor *);
   void removeEditor(char *name);
   void send(char *editorName, Event e);

};
```

- An `Editor` class is an interface for the `TextEditor` and `ShapeEditor` classes. Various manipulations can be applied on an editor which are cut, copy, paste, close and file saving. Each event is handled and fulfilled by the `handle` method.

`copy` method keeps the selected part of the editor (the interval of the indexes of the selected string in the text for `TextEditor` and selected Geometric Object in a `ShapeEditor`) in memory to paste in the future.

The select method for `TextEditor` and `ShapeEditor` are particular for themselves and does not have a signature in `Editor` class.

`cut` method keeps the `GeometricObject or char*` (or character number after/before cursor) in memory while it removes the object from the editor.

`paste` object pastes the copied or cut `GeometricObject` in (0,0) point for `ShapeEditor` and char * starting from the index represented by the Cursor.

`draw`() method outputs the contents of the editor; it prints the text of in the Text `Editor` or gives the descriptions of the `GeometricObjects` in the `ShapeEditor`.

`deleteObject` method removes the selected object (substring in text or GeometricObject) with the event DELETE.

`save` method overwrites the object features to the lastly written file. It saves the features of an editor into the file with

the same name as the editor itself.

```
class Editor {

protected:

   char *editorName;

public :

   Editor();

   virtual void save()=0;

   virtual void copy()=0;

   virtual void cut()=0;

   virtual void paste()=0;

   virtual void deleteObject()=0;

   virtual void draw()=0;

   virtual void handle(Event)=0;

}
```

- A `TextEditor` class should include `addText`, `replace` and `select` methods to collect the texts added into the editor, to replace a string with another for all occurrences and select a string according to the location of the cursor respectively.

`TextEditor(char *name)` constructor should be called with the char* argument to define it by its name.

`void addText(char *)` method adds the string in the argument starting from the index of the Cursor. After each insertion of text, the cursor should point to after the last character index inserted. That means an insertion can be made following the last character in the text or in the middle of the text depending on the cursor index. Some samples are:

Assuming the cursor shows 0^{th} index at the very beginning;

- addText(“0123”); // the cursor shows 4 th index which is null

0123^

- addText(“45”); // the cursor shows 6^{th} index

012345^

- moveCursor(4);

0123^45

- addText(“abc”) ; //the insertion is made starting from the 4^{th} index and the cursor shows 7^{th} index

0123abc^45

- moveCursor(9);

0123abc45^

- select(-3);

- deleteObject();

0123ab^

- moveCursor(1);

0^123ab

- select(2);

- deleteObject();

0^3ab

`char * replace(char *, char*)` method replaces all of the occurrences of the first argument with the second argument string.

`void select(int c)` method selects "c" many characters starting from the cursor position. If c>0, then c many characters after the cursor should be selected, if c<0, then c many characters before the cursor should be selected. If the number c exceeds the last character when c>0 (or first character when c<0) then the selection ends with the border. After the selection, the cursor position is updated to the last character index.

Every `TextEditor` should include a `Cursor` member to point to any character in the text.

Cursor& getCursor() method should return the reference to the cursor member.

- A `Cursor` class should include `moveCursor` method to change the cursor location to the desired index:

```
void moveCursor(int newPos);
```

The initial point of a cursor is 0 by default:

```
Cursor(int a=0);
```

- `ShapeEditor` class may include any number of `GeometricObjects` as stated before. The constructor should include the editor name as argument `ShapeEditor(char *)`.

`void addGeometricObject(GeometricObject *)` method should add one of `Rectangle`, `Triangle` and `Circle` objects into the editor .

GeometricObject* pick(Point) selects the **last** object whose area includes the selected point in the editor by returning the selected object. If a point which is not in any object bound is selected then the selected object in memory (if there are any) is deselected.

- A `GeometricObject` class includes the methods which are used to make modifications on the size and coordinate of the objects which are `scale`, `move` and `rotate`. `move` method changes the coordinate of the object by its center point. Rotation is done by giving the angle and direction parameters. The angle is assumed to be in

degree type. `scale` method changes the size of the object without changing the center point. So it should also change the given edge points for triangle and rectangle and the radius of circle. `area()` method is also calculates the area particular to the geometric object type. Every `GeometricObject` has a center, angle and color defined. The default values for center is (0,0), angle is 0.0 and color is WHITE.

```
class GeometricObject{

protected:

Point *center;

double angle;

Color color;

public:

  GeometricObject();

virtual double area()=0;

virtual void scale(int scalefactor)=0;

virtual void move(Point)=0;

virtual void rotate(double, Direction) { };

void setColor(Color);

}
```

- `Circle` class instance is constructed by its radius along with the center point `Circle(Point, double radius)`

- `Rectangle` class instance is constructed by the two diagonal points (bottomleft and rightop in order): `Rectangle(Point, Point)`

- `Triangle` class instance is constructed by the three edge points: `Triangle(Point p1, Point p2,Point p3)`. Since the angle for a newly created object is 0, you can assume that p1 defines the upper edge, p2 defines the bottom left and p3 is the bottom right edge points.

- A sample output for the draw() call for a TextEditor is the content in the editor along with the cursor position such as:

A sample text to define t^he output of a TextEditor.

where the cursor is represented by ^ and it points to 'h' character.

- A sample output to define the content of a ShapeEditor is:

Rectangle [center] [angle] [color] [Point1] [Point2]

Triangle [center] [angle] [color] [Point1][Point2] [Point3]

Circle [center] [color] [radius]

Specifications:

- You will have two files. One is **hw5.h** for your class declarations, structures, enumerated types; the other is **hw5.cpp** for your definitions. You should not write main function in these files. I will write a main function in hw5main.cpp to test your codes (a simple one is provided with the homework). If you want to test your codes before submitting, write a main function in hw5main.cpp and use Makefile to compile your codes on inek machines.

- Follow the newsgroup for any questions and possible updates.

- All the work should be done **individually**.

- You will submit a single tar file **hw5.tar** including **hw5.h**, hw5.cpp and **Makefile**. You can tar your files with the command "tar cvf hw5.tar hw5.cpp hw5.h".

- You will submit your codes through **cow** system. Specifications (file name, method names, class name, types etc.) are strict.