Project 3 Specs

Goals

- Gain more experience managing the data in a document
- Gain experience saving and opening documents in a graphical user interface

Description

In this project you will have to implement a file-based document for your Project 2 application. The new functionality should support creating new documents, loading and saving documents from storage, and marking the documents as "dirty" when they are modified.

Document

Documents are often implemented by the New/Load/Save commands.

- **New** - creates a new window (new Project3()) and connects a document (empty initially) to the window. When the user attempts to save the document for the first time, the Save dialog box will be shown.
- **Load** - reads a file from storage and loads the data into the currently opened window/document.
- **Revert to saved** - replaces the document data with the version of the data stored in storage. Note that this is only available after the document has been saved.
- **Save** - saves the document data to storage, prompting the user to name the document and select the folder/directory where to save the data.

From Project 2, your have an application that manipulates different types of shapes. For Project 3, we are going to extend Project 2 to support saving and opening documents.

The project below explains the file format for the document. We are using a text-based file format for the project. This should make your code simple as you won't have to parse complicated files. But in addition, you should be able to save files and share them with others in class so they can load your files in their programs.

Shapes syntax

Your program should read/write the following shapes to storage in text format.

- Circle, Rectangle, Triangle, Square, and Line

In addition, and for each shape, it should write the color for filling the shape (if it is filled) and the color for the border for the shape.

Fortunately, the SShape class has a toString() method that will produce a text representation of shapes in a format that should be appropriate to write to file. In addition, all SShape have constructors that initialize them using a text representation that matches part of the text representation generated by toString(). The constructor does not, however, process the colors from the file. That you must process separately.

Below is the content of a file (example.txt) that shows the commands for all the shapes.

```
# example from project3

# circle
border 0 0 0
fill 255 255 0
circle 107 98 50
```

```
# rectangle
border 0 0 0
fill 255 102 102
rectangle 22 19 175 28

# triangle
border 255 102 102
fill 217 72 190
triangle 103 108 53 208 153 208

# line
border 0 0 0
line 211 21 210 207
```

===

Thuat toan

Phan tich tung dong line tren file data

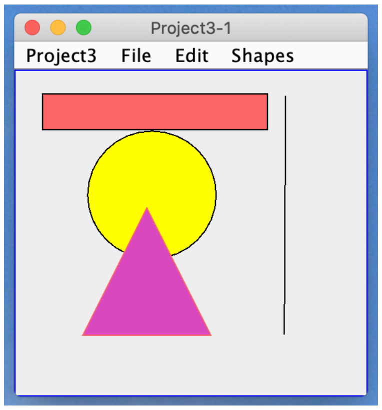
If token la keyword co trong keyword dictionary thi xu ly dua tren token la keyword nao. If token la border then xu ly cho border. If token la fill thi xu ly cho fill. If token la rectangle thi xu ly cho rectangle va add vao list of shapes trong truong hop token thuoc list of shape {circle, rectangle, triangle, square, line}.

Init two object types that are border and fill with three attributes.

keyWordList = {“border”, “fill”,”...}

If a token belongs to the list of keywords, process code according to keyword and if token belongs to list of shapes then create object and add to list of shapes.

For reference the image below shows the graphical display of this file.



File format

The file format has a line per object. A line can be either a comment of a textual description of the object type. Attributes (fill color and border color) are described in a separate line, as seen above.

- Comments - lines that begin with a # in the first character of the line are to be ignored. Blank lines also to be ignored.
- border 0 0 0 the integers following the border command are the three values for the Red, Green, and Blue for the color.
- fill 255 102 102 like the border command, is followed by 3 integers representing a color.
- circle 107 98 50 represents a circle with center at 107,98 and with radius 50.
- rectangle 22 19 175 28 denotes a rectangle with top left corner at 22,19, width of 175 and height of 50.
- square 10 10 30 is a special rectangle that has the top left corner at 10,10 and with width and height of 50. Note that there is no corresponding class of SSquare, you must create an SRectangle. It is ok to display it as a rectangle that looks like a square and ok to allow the user to resize it. When you save it, it is ok to save it as a rectangle. This shape type only exists for input from a file.
- triangle 103 108 53 208 153 208 represents a triangle with the three corners at p1=103,108, p2=53,208, and p3=153,208.
- line 211 21 210 207 represents a line from 211,21 to 210,207.

Functional Requirements

A. SDocument

The SFramework already provides most of the behavior needed to implement documents. If you take a look at the document examples you will see that the basic behaviour is provided. You need to provide the behavior for the parts that are unique to your project.

Create your Project3 as a class that extends SWindow, as you have done with the previous projects. For this project, initialize your class this way:

```
public Project3()
{
  super("Project3");
}
```

Create an extension of SDocument and attach it to the Project3 by calling attachDocument(). It might be easiest to do it through an anonymous class.

You must implement these three methods:

```
public void init();
public void write(File file) throws IOException;
public void read(File file) throws IOException;
```

1. write(File file) the write method should write to a text file using the file reference passed as argument (and also stored in the itsFile property). The data written should follow the format described above.
   Writing the file is nothing more than building a loop over your list of shapes and writing to the file the string value returned by toString() for each shape.

Need to write code for processing the different types of shapes list

2. read(File file) the read method should read a text file from the file reference passed as argument (and also stored in the itsFile property).
I strongly recommended that you use a Scanner to parse the input. This example reads each line of the file.

```
try (Scanner sc = new Scanner(file)) {
        while (sc.hasNextLine()){
                System.out.println(sc.nextLine());
                // do something with the line
        }
}
```

```
catch (IOException e) {
        e.printStackTrace();
}
```
The line of input from the file, can itself be split by a new scanner. Here is an example:
```
Scanner scanner = new Scanner(inputLine)
scanner.useDelimiter("\s");
while (scanner.hasNext()) {
        Str cmd = scanner.next();
        int x = scanner.nextInt();
        ...
        2.   }
```

B. Project3

1. If the user clicks on an object (selects it) or moves an object, make sure you mark your document "dirty" by calling markDirty().

C. Project3 (from project 2)

1. You were supposed to add this to Project2. Just repeating it so that grading works correctly. Add a method public int numShapes() to your Project2 class. This routine should return the number of shapes you have in your ArrayList list.
2. In addition, add public ArrayList<SShape> getShapes() to your project 3 and have it return the array list where you store your shapes. Again, needed fo grading.

Submission

You must submit to Web-CAT all your project 3 files required to run the project.

You do not need to submit the SFramework.jar. Do not submit data files. All of those files are already stored in Web-CAT.

1. Document your code, it makes it easier to read, easier to grade, and easier for you when you revisit it for future projects. Follow the CCI Style Guide for Java.
2. Do not put your code in any user-defined packages (you can use existing Java packages). The projects in this class are not big enough to merit the use of packages. Using packages in these projects complicates the testing unnecessarily. Note this is not true for commercial software development, but certainly true for me trying to grade your code and finding that everybody used a different name for the main package.
3. Stick to the names mentioned here. Following the right naming conventions is essential for me to be able to give you feedback quickly. In particular, the names of the menu items must be just as shown here, the testing code will look for those menu items by name and if you name them differently, it will cause the testing program to fail, thus delaying feedback.

TLDR

- Project 3 Assignment (video available in the assignment page)
- Sample Data Files

Need to check with all example file below to make sure the program is good (cci.txt)

  - cci.txt
  - example.txt
  - cci-bw.txt
  - cci-colors.txt
- Coding
  - CCI Style Guide for Java

    - JGrasp
    - (Links to an external site.)
    -
    - JGrasp Video for 4440
- SFramework
    - Documentation
    - Video with examples of SFramework
    - Document (PDF) with examples of SFramework
    - Browse the Javadoc for SFramework

the end