Scaled Picture Lab

Overview
In this lab you will learn how to use some of the basic Java2D Graphics features to create color paintings composed of rectangles, ovals, circles, and lines. To make sure the images can be shown in any size and shape, you will learn the basics of linear scaling (transformations) in two dimensions. Finally, you will use a class that extends another class and learn the basic facts about extending classes and using the derived classes.

Note about terminology: The class that contains the basic information is called the base class. There can be several different classes that extend the base class. These are called derived classes.

Program Summary
The program you will work on has a large number of controls, though you are already familiar with many of them. The controls are organized in two columns, and the graphics window is to the right of all controls. We will show the snapshot first:

The snapshot of the ScaledPicture program is showing four different pictures. Tree and Sun picture is shown with a grid to help in deciding the coordinates of the features.

The left part of the controls allows you to select one of six different paintings or images to display and shows the title of the picture and the name of the author in the Picture Title
annotations. Underneath are the key picture drawing controls. These allow you to show the picture, show a grid over the picture in a manner similar to that used in the Turtle Explorer Lab, reset the picture state - as was done in the Picture Explorer Lab, and clear the graphics window. You are already familiar with the functionality of all these controls. The only new button is one with the label Picture Mosaic. Currently, it does not do anything. Your last task will be to program this button to create a spectacular mosaic of the six paintings and images.

The right part of the controls is similar to the controls you used in the Picture Explorer Lab. You can see and can change the location of the picture and its width and height. Underneath are two sets of buttons for testing your picture. The pre-programmed buttons display several copies of the picture is four different arrangements - four pictures tiling the whole window, a mosaic of 3 rows and 6 columns, a mosaic of 6 rows and 4 columns, and finally five pictures across the main diagonal. The last control panel lets the user select the number of desired columns and rows to show.

This is a snapshot of 4 x 6 mosaic of Locomotive pictures:

![Image of 4 x 6 mosaic of Locomotive pictures]

**Lab Tasks**

Your tasks will be the following:

1. Study the code for creating the Tree and Sun drawing to learn about the Java graphics functions and also about extending classes.
2. Implement the `showPicture` function for the Face class to display the shown face.
3. Create a new `studentPicture` class that will build a picture that you design.
4. Write the code that will implement the drawing of the user selected mosaic (with the given number of columns and rows).

5. Write the code that will implement the Picture Mosaic action to display a combination of at least two different pictures in some arrangement.

Overview of the source code

The snapshot of the project shows you which Java files you will need. You will actually create a completely new file for the fourth picture and add it to the project.

The `ScaledPictureApplication` is the class that builds the GUI, and includes all action functions. You will need to implement the two action functions that correspond to the **Rows and Columns** button and to the **Picture Mosaic** button. Function headers are already included. In addition, you will need to modify the function that corresponds to the **Student** options button that selects which picture will be used. Currently it shows the **Locomotive** picture, but when you are done, it will show the picture you designed.

The `ScaledPicture` class is an abstract base class for the four classes that define real pictures. (More about the abstract stuff later).

Two of the picture classes `TreeAndSun` and `Locomotive` are complete - they are included as examples that you can study and emulate.

The third picture class, `ScaledImage`, is designed to display any given .gif image as a scaled picture, and is used to display both the `Flowers.gif` image and the `DoverCastle.gif` image.
The Face class is missing the body of the showPicture function. Your job will be to build this function so it shows your picture.

Finally, you will build one class from scratch. The only reason why we insist that you call this class StudentPicture is to make it easier on the grader to run all your projects.

**Task 1: Code Reading**

1. Study the code for creating the Tree and Sun drawing to learn about the Java graphics functions and also about extending classes.

You should start by reading the ScaledPicture.java code. The first thing you notice is that the class header calls this an abstract class. This means, that we did not complete the class definition - something is missing and so we cannot build an object that is an instance of this class.

The main part of the class - the constructors, and the set and get functions should look familiar by now - they define a class very similar to the one you worked with in your Picture Explorer Lab.

Answer the following questions about this class:

- **SP-1** What are the member data for this class?
- **SP-2** How many constructors are there in this class?
- **SP-3** What information is included here that was not used in the Picture Explorer Lab?
- **SP-4** How many set and get functions are there?
- **SP-5** Show the headers of all other member functions in this class (not the constructors, not the set and get functions).

The last function in this class contains only the function header:

    public abstract void showPicture(BufferedPanel window);

Because every picture will be different, we need to define a different showPicture function for each picture we want to show. One way to do this is by creating a new class for each picture, that extends the ScaledPicture class, and includes only the definition of this function (and any helper function we may need).

The class TreeAndSun shows how this is done. You can see that it is a very short class. It starts by informing the compiler that this class extends the ScaledPicture class. This means, any object in this class will have all the public and protected member data defined in ScaledPicture and can call all the public and protected member functions in that class. We only need to do two things here.

First, we need to specify the constructor. The compiler does not assume that we would be happy to use the constructor in the base class - we have to say so. The constructor with two arguments:
public TreeAndSun(String title, String author){
    super(title, author);
}

consists of one function call super(title, author); This is a call to a similar constructor in the ScaledPicture class. Such call must be done at the start of the constructor function, though we can follow with additional work afterwards. We will see this in ScaledImage class.

The rest of the class is the definition of the showPicture function and a helper function that returns the minimum of two numbers.

We now look at how the picture is made. Please, start by reading the Graphics in Java short tutorial.

Next snapshot shows the Tree And Sun picture, full size with a grid to help you in studying the code:

The code for showPicture function starts with defining the Graphics2D object, and the Rectangle, Ellipse, and Line objects. We then paint each feature, understanding that a new feature will be painted over those already painted. This is useful when painting the tree. We first paint the tree trunk - a bit higher than needed, then cover it partially with tree top.

We explain here how the tree is painted - leaving the rest as questions for you to answer. We first paint the tree trunk. Its width is 1/10 of the total width, its height is 4/10 of the...
height. The top left corner is hidden - its location is 6/10 of the width to the left of the left
edge and 6/10 of the height down from the top edge. This leads to the following code:

```java
// the tree trunk: 1/10 wide, 4/10 tall
G.setPaint(new Color(100, 100, 20));
R.setRect(x + 0.6 * width, y + 0.6 * height,
0.1 * width, 0.4 * height);
G.fill(R);
```

The tree top is an ellipse framed by the rectangle of the width equal to 1/2 of the total
width and height equal to 6/10 of the total height, with the top left corner located at
position \((x + 0.4 * width, y + 0.1 * height)\).

```java
// the tree top: a bit different shade of green
G.setPaint(new Color(100, 255, 20));
E.setFrame(x + 0.4 * width, y + 0.1 * height,
0.5 * width, 0.6 * height);
G.fill(E);
```

The only other interesting feature in this picture is the sun. To make sure it does not get
too big, we derive its size as 1/10 of the smaller of the two dimensions (width and
height). The ellipse then has the same width and height - making it a circle.

Answer the following questions about the rest of the code:

**SP-6** What are the coordinates of the top left corner of the frame for the sun?

**SP-7** Sketch a drawing of the V-shaped bird figure and show what are the coordinates
of the end points that define the two lines.

**SP-8** Read the code to find out the height and the width of the rectangle that paints the
sky. Explain why this is possible.

### Task 2: Create a given picture

2. Implement the `showPicture` function for the `Face` class to display the shown face.

The snapshot on the next page shows the face painted in four corners of the window, the
last picture painted with a grid to help you in designing your painting. Start by listing the
features of the picture and identify the rectangles, the ellipses, the circles, and the lines
that you will need to make each feature. For each shape, specify the top left edge of the
frame and its width and height, as we have done above. Of course, for lines, there are
only two end points, there is no width or height. The colors for each feature are given.

Use the following format and include this in your report (F-1):

<table>
<thead>
<tr>
<th>Feature:</th>
<th>Background</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape:</td>
<td>Rectangle</td>
</tr>
<tr>
<td>Color:</td>
<td>gray: Color(100, 100, 200)</td>
</tr>
<tr>
<td>left:</td>
<td>= x</td>
</tr>
<tr>
<td>top:</td>
<td>= y</td>
</tr>
<tr>
<td>w:</td>
<td>width</td>
</tr>
<tr>
<td>h:</td>
<td>height</td>
</tr>
</tbody>
</table>
Add the code for painting each shape to the `showPicture` function in the class `Face`. You can test your program after you have added each feature. Make sure you try both 3 x 4 Mosaic and 6 X 4 Mosaic - to see if the picture stretches in each direction.

**Task 3: Create a new picture**

3. Create a new `StudentPicture` class that will build a picture that you design.

This task is similar to what you did in the Turtle Explorer Lab. Use your creativity to design some interesting picture that contains at least two rectangles, two ellipses, two lines and one circle. Document your design in your lab report - first showing a sketch of your drawing on a graph paper, then following by a list of features as you have done for the `Face` picture.

Now you can create a new class! Copy the `Face` class as given in the original lab - with nearly no code in the `showPicture` function. Rename the file in the Windows folder to be `StudentPicture.java`. Now open the file while your MetroWerks project is open. In Project menu select *Add StudentPicture.java to Project*. The file is now a part of the project.

Now you need to modify the file, so it contains code for `StudentPicture` class, not `Face` class. Find every place where `Face` appears in this file and replace it with `StudentPicture`. (There are four places in this file where you need to do this.) Copy the lines you needed to change into you report.

To make sure you can actually see your picture you need to modify slightly the `ScaledPictureApplication.java` file. Find the function `studentPicture()`. This is the
function that is performed when you press the Student option button. The function now looks as follows:

```java
public void studentPicture(){
    myPicture = new Locomotive("Locomotive", "Professor Proulx");
    refresh();
}
```

Obviously, you do not want to paint the Locomotive picture and you certainly want to get credit for your work. So, enter the name of your picture and your name in the appropriate places. Do not make the picture name any longer than Locomotive - otherwise, you will have to resize the graphics window to view the picture properly.

Finally, you are ready to write the code for your picture. For each feature in your picture, add the code, including the changes in color, to the body of the showPicture function. Make sure you include comments, specifically - make sure you describe the picture at the start and identify yourself as the author, then identify each feature as has been done in the TreeAndSun class.

Include the printout of your StudentPicture.java file with your lab report together with a full size snapshot with the grid showing too.

**Task 4: Implement user selected mosaic**

4. Write the code that will implement the drawing of the user selected mosaic (with the given number of columns and rows).

The body of the rowcol() function in the ScaledPictureApplication class currently contains only a call to a function refresh(), that resets the views of all member data for the current picture to its proper values. So, if the last picture shown will be in the bottom right corner, its location and size will be shown in the GUI.

Your code should start in reading the user supplied choices of the number of rows and columns. Adjust these values, so that if the number of rows or columns is less than or equal to one, then the selection will be changes to 1.

Determine the size of the picture (width and height) and set the size and width accordingly. Now write a double loop that will show the picture as desired. You will need to keep setting the location of the picture before you show it.

Copy the code for this function into your lab report and include a snapshot of a 4 x 7 mosaic of your picture.

**Task 5: Be creative**

5. Write the code that will implement the Picture Mosaic action to display a combination of at least two different pictures in some arrangement.
The last task gives you another chance to be creative. The function `mosaic()` is called when the user presses the button **Picture Mosaic**. Right now it only clears the window.

```java
public void mosaic(){
    // clear the graphics window
    window.clearPanel();
    // Create a mosaic of the six images - combining different sizes
    // pictures, shapes
    window.refresh();
}
```

Create at least two objects in the derived classes of the `ScaledPicture` class (`TreeAndSun`, `Locomotive`, `Face`, `StudentPicture`, `Castle`, etc.). Write the code - including possible loops that will show a mosaic of pictures in the graphics window.

Here is a sample of a mosaic of locomotives, trees, and flowers:

![Mosaic Example](image.png)

Copy the code for this function into your lab report and include a snapshot your mosaic.

Submit electronically the two Java files you worked on: `StudentPicture.java` and `ScaledPictureApplication.java`, as well as your lab report.