WORKED EXAMPLE 2.2 Working with Pictures

Problem Statement Edit and display image files in the `Picture` class found in the `ch02/worked_example_2` directory of this chapter's companion code.

For example, the following program simply shows the image given below:

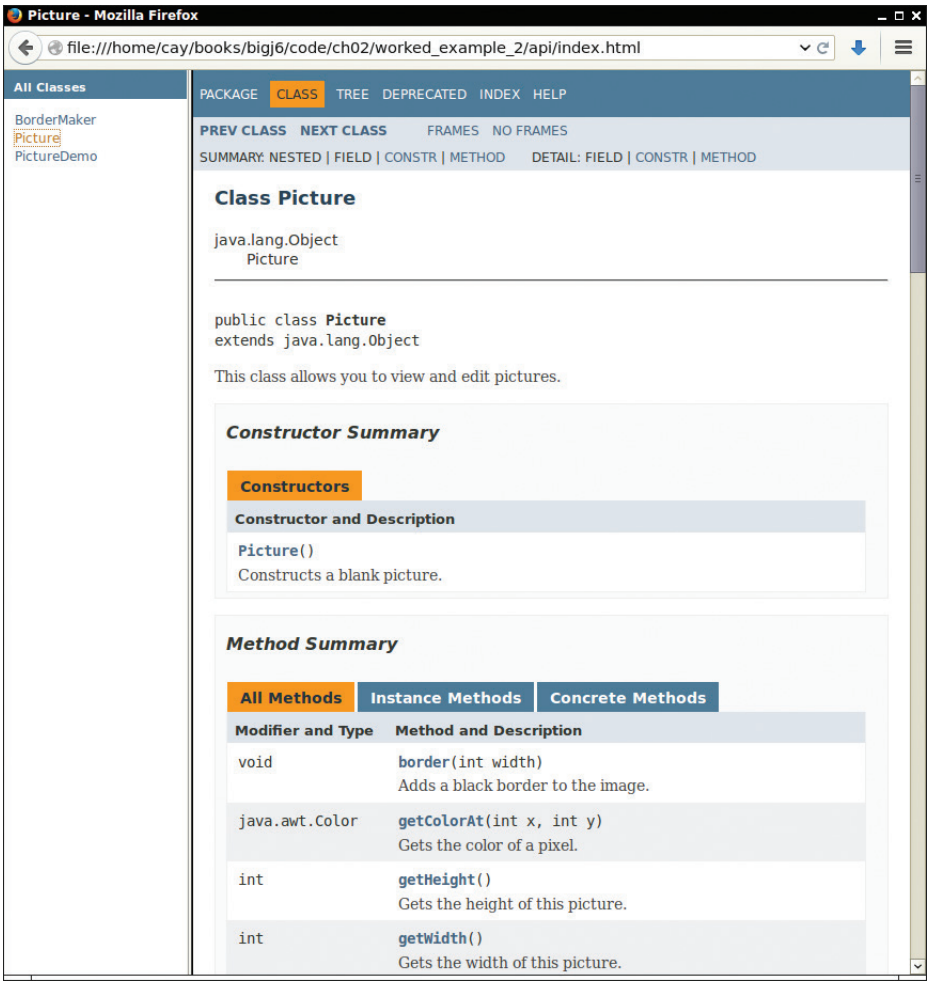
```
public class PictureDemo
{
   public static void main(String[] args)
   {
      Picture pic = new Picture();
      pic.load("queen-mary.png");
   }
}
```

Cay Horstmann.

Your task is to write a program that reads in an image, shrinks it, and adds a border. Shrink it sufficiently so that there is a transparent border inside the black border, as in the figure below.

Cay Horstmann.

You should *not* look inside the internal implementation of the `Picture` class. Instead, use the API documentation by pointing your browser to the file `index.html` in the `worked_example_2/picture/api` subdirectory.



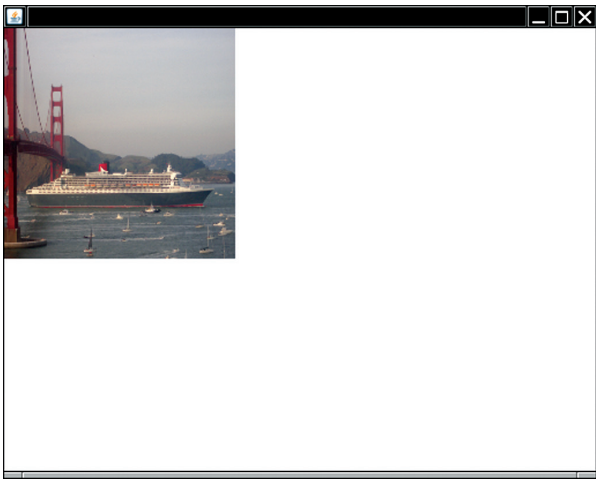
The API contains a number of methods that are unrelated to the task, but two of the methods are clearly useful:

```
public void scale(int newWidth, int newHeight)
public void border(int width)
```

If the method comments are not clear, it is a good idea to write a couple of simple test programs to see their effect. For example, this program demonstrates the `scale` method:

```
public class PictureScaleDemo
{
   public static void main(String[] args)
   {
      Picture pic = new Picture();
      pic.load("queen-mary.png");
      pic.scale(200, 200);
   }
}
```

Here is the result:



Cay Horstmann.

As you can see, the picture has been resized to a 200 × 200 pixel square.

That's not quite what we want. We want the picture to be a bit smaller than the original. Let's say that the black border is 10 pixels thick, and we want another transparent border of 10 pixels. Then the target width and height are 40 pixels less than the original, leaving 20 pixels on each side for the borders.

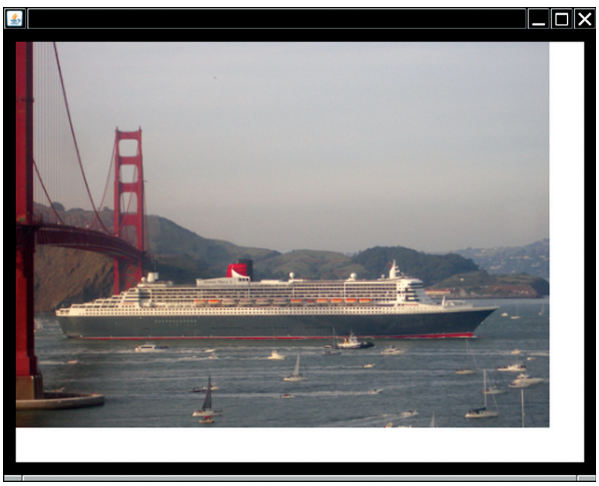
Looking at the API, we find methods for obtaining the original width and height. Therefore, we will call

```
int newWidth = pic.getWidth() - 40;
int newHeight = pic.getHeight() - 40;
pic.scale(newWidth, newHeight);
```

Then we add the border:

```
pic.border(10);
```

The result is



Cay Horstmann.

If we can move the picture a bit before applying the border, we are done. Another look at the API reveals a method

```
public void move(int dx, int dy)
```

That's just what we need. The picture needs to be moved 20 pixels down and to the right. Our final program is

worked_example_2/BorderMaker.java

```
1  public class BorderMaker
2  {
3     public static void main(String[] args)
4     {
5        Picture pic = new Picture();
6        pic.load("queen-mary.png");
7        int newWidth = pic.getWidth() - 40;
8        int newHeight = pic.getHeight() - 40;
9        pic.scale(newWidth, newHeight);
10       pic.move(20, 20);
11       pic.border(10);
12    }
13 }
```

Couldn't we have achieved the same result with an image editing program such as Photoshop or GIMP? Yes, but it is an easy matter to extend this program so that it can automatically apply a border to any number of images.