Project 03: Making AND using methods that blend (a) WHITE and (b) BLACK with a rectangle GIVEN BY PARAMETER VALUES with an amount from 0.0 to 1.0 GIVEN BY ANOTHER PARAMETER VALUE

Methods MUST be testable by OUR EXISTING APPLICATION CODE!!!
5.2.3 Blending Pictures
When we create collages by copying, any overlap typically means that one picture shows over another. The last picture painted on is the one that appears. But it doesn't have to be that way. We can blend pictures by multiplying their colors and adding them. This gives us the effect of transparency.

We know that 100% of something is the whole thing. 50% of one and 50% of another would also add up to 100%. In the program below, we blend a picture of the two sisters with an overlap of 50 (the width of Katie minus 150) columns of pixels (Fig 5.12) onto the current picture.” G&E Sec 5.2.3
Program 28: Blending Two Pictures

/**
 * Method to blend two sisters together onto this, the current Picture.
 */

public void blendPictures( )
{
    Picture katiePicture = new Picture( something );
    Picture jennyPicture = new Picture( something else );
    Pixel katiePixel;
    Pixel jennyPixel;
    Pixel targetPixel;

    //There are 3 doubly-nested loops. In the body of each loop,
    //the pattern of code is
    katiePixel = katiePicture.getPixel( some x-location, some y-location );
    jennyPixel = jennyPicture.getPixel( some x-location, some y-location );
    targetPixel = this.getPixel( some x-location, some y-location );

    targetPixel.setColor ( some Color determined from Colors of katiePixel, jennyPixel or both. );
Plan

1) 1) A couple of Albany improvements

2) How each blended Color is computed, applying AND coding calculations that
   MIX
   int (integer) whole numbers
   WITH
   double (floating point) numbers with fractional parts
Program 28: Blending Two Pictures

/**
 * Method to blend two sisters together onto this, the current Picture.
 */

public void blendPictures(String filename1, String filename2)
{
    Picture katiePicture = new Picture(filename1);
    Picture jennyPicture = new Picture(filename2);
    Pixel katiePixel;
    Pixel jennyPixel;
    Pixel targetPixel;

    //There are 3 doubly-nested loops. In the body of each loop,
    //the pattern of code is
    katiePixel = katiePicture.getPixel(some x location, some y location);
    jennyPixel = jennyPicture.getPixel(some x location, some y location);
    targetPixel = this.getPixel(some x location, some y location);

    targetPixel.setColor(some Color determined from Colors of katiePixel,
    jennyPixel or both.);
Program 28: Blending Two Pictures
/**
 * Method to blend two sisters together onto this, the current Picture.
 */
public void blendPictures(String filename1, String filename2) {
    Picture katiePicture = new Picture(filename1);
    Picture jennyPicture = new Picture(filename2);
    Pixel katiePixel;
    Pixel jennyPixel;
    Pixel targetPixel;
    G&E initialize these 3 variables to null, which
    (A) doesn't affect anything, 
    (B) is unnecessary, 
    (C) is silly programming, 
    and (D) misleads since they are always assigned to 
    later, for each Pixel!
// Method to blend two sisters together onto this, the current Picture.

public void blendPictures() {
    Picture katiePicture = new Picture(something);
    Picture jennyPicture = new Picture(something else);
    Pixel katiePixel;
    Pixel jennyPixel;
    Pixel targetPixel;

    // There are 3 doubly-nested loops. In the body of each loop,
    // the pattern of code is
    katiePixel = katiePicture.getPixel(
        some x-location, some y-location);
    jennyPixel = jennyPicture.getPixel(
        some x-location, some y-location);
    targetPixel = this.getPixel(
        some x-location, some y-location);
    targetPixel.setColor (
        some Color determined from Colors of katiePixel, 
        jennyPixel or both.);
katiePixel =
    katiePicture.getPixel( sourceX, sourceY );
jennyPixel =
    jennyPicture.getPixel( sourceX - 150, sourceY );

targetPixel =
    this.getPixel( targetX, targetY );

targetPixel.setColor(
    new Color( (int) (katiePixel.getRed() * 0.5 + jennyPixel.getRed() * 0.5),
                (int) (katiePixel.getGreen() * 0.5 + jennyPixel.getGreen() * 0.5),
                (int) (katiePixel.getBlue() * 0.5 + jennyPixel.getBlue() * 0.5) ));
Weighted average calculation programmed.

\[
\text{new Color( (int) }
\begin{align*}
(katiePixel\text{.getRed()} & \times 0.5 + \\
jennyPixel\text{.getRed()} & \times 0.5)
\end{align*}
\]

with weight \( w_1 \) equal to 0.5 and weight \( w_2 \) equal to 0.5
Java's precedence rule: “* is stronger than +” makes the computer do the two multiplies FIRST, before doing the + (LAST) in

```
katiePixel.getRed() * 0.5 + jennyPixel.getRed() * 0.5
```

It's the same old rule from math class.
katiePixel =
katiePicture.getPixel( sourceX, sourceY );
jennyPixel =
jennyPicture.getPixel( sourceX - 150, sourceY );
targetPixel = this.getPixel( targetX, targetY );
targetPixel.setColor(
    new Color( (int) (katiePixel.getRed() * 0.5 + jennyPixel.getRed() * 0.5 ),
                (int) (katiePixel.getGreen() * 0.5 + jennyPixel.getGreen() * 0.5 ),
                (int) (katiePixel.getBlue() * 0.5 + jennyPixel.getBlue() * 0.5 )
    );
Heed The ANYTHING with ANYTHING principle in Java's language design
Proj03: YOU figure out code for programming the blending of BLACK [(r,g,b) equals (0,0,0)] and WHITE [(r,g,b) equals (255,255,255)] WITH the Color from a Pixel in this Picture, with weights=(amount) and (1.0-amount) For the extra credit, blend (r,g,b)'s from a Color parameter WITH the Color from a Pixel in this Picture (and apply that extra work effectively.)
Proj03: YOU figure out code for programming the blending of BLACK \([(r,g,b) \text{ equals } (0,0,0)]\) and WHITE \([(r,g,b) \text{ equals } (255,255,255)]\) WITH the Color from a Pixel in this Picture, with weights\(=\)\((\text{amount})\)\(\text{and}(1.0-\text{amount})\)

For the extra credit, blend \((r,g,b)\)'s from a Color parameter WITH the Color from a Pixel in this Picture (and apply that extra work effectively.)
Now, after you figure out what to program in the ( ... ) to compute each weighted average, we explain how to convert the resulting three doubles into ints.
new Color(
(int) ( katiePixel.getRed() * 0.5 +
jennyPixel.getRed() * 0.5 ),
(int) ( katiePixel.getGreen() * 0.5 +
jennyPixel.getGreen() * 0.5 ),
(int) ( katiePixel.getBlue() * 0.5 +
jennyPixel.getBlue() * 0.5 )
);

Calculate with decimals (fractions) FIRST.

These ( .. ) mean do THAT calculation FIRST.

AND ALSO, do the cast operation SECOND.
Good

setRed(int) DEMands an int parameter value!

That 117.0 parameter value has type double, not int!

BUT-There's a 2nd syntax error!
(int) [the cast operator] is STRONGER THAN * [the multiply operator]!
First, (int) 234 leaves 234 unchanged. Then ...
Second, 234*0.5 is 117.0 has type double.
setRed( 117.0 ) is illegal, since setRed DEMANDS an int parameter value!

So.. you NEED to HEED our instructions, and code (int) (your double calculation) (double)
How to convert a while loop into a for loop.

while( TEST )
{
    STATEMENTS;
    TO;
    REPEAT;
}

for( INIT.; TEST; UPDATE )
{
    STATEMENTS;
    TO;
    REPEAT;
    }

UPDATE ;
while( ) {
    
    for( ; ; ) {
        
        INIT.
        STATEMENTS;
        TO;
        REPEAT;
        
        TEST
        STATEMENTS;
        TO;
        REPEAT;
        
        UPDATE
        ;
    
} 

1. Parts are **done** in the order they are written (usually).
2. Easier to get the logic right.
3. The INIT. stuff can be several statements.
4. The loop is not expressed as a self-contained thing.

1. It's hard for beginners to analyze the sequence exactly right.
2. The 3 parts (usually) needed for loop control are in the same line.
3. The 3 part for( ; ; ) reminds us to code: (1) INITIALIZATION (2) TEST (3) UPDATE (increment or decrement.)
4. Declarations in INIT. are local to the loop.
A nested loop is a loop, called the outer loop, whose statements to repeat comprise a loop, called the inner loop.

The group of statements to repeat IS a loop...the group is a loop INSIDE a loop!
while TEST
{
  INIT.
  ;  
  while TEST
  {
    INIT.
    ;
    STATEMENTS;
    TO;
    REPEAT;
    UPDATE
    ;
  }
  UPDATE
  ;
}
When the inner loop is a for loop, you can easily see the inner loop as a SELF-CONTAINED thing.
Thinking about nested loops.

- FIRST, think of what the whole inner loop does, in a single thought. For example, “paint a row of Pixels.”
- SECOND, think that the outer loop repeats the single thing the inner loop does: Paint many rows of pixels to fill in a rectangle.

- FIRST, think of how many repetitions the outer loop makes, and what they are for. For example: “repeat once for each row”
- SECOND, think of what the whole inner loop does in a single thought.