Name: ______________________________

Date: _______________

ACTIVITY 3

Identifying a Hidden Picture

1. In the `Steganography` class, after modifying `canHide` to allow the secret image to be smaller than the source image, write a second version of the `hidePicture` method that allows the user to place the hidden picture anywhere on the modified picture. This means adding two parameters for `startRow` and `startColumn` that represent the row and column of the upper left corner where the hidden picture will be placed.

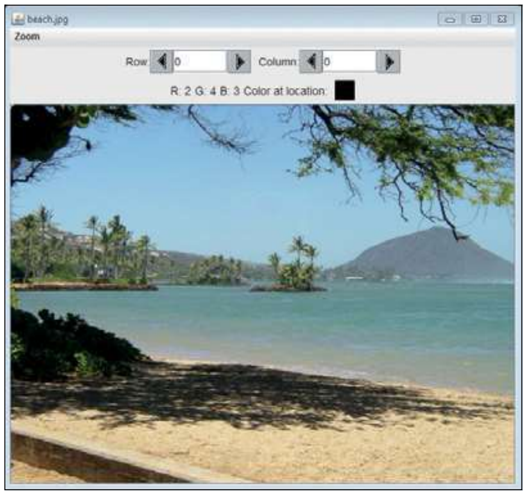
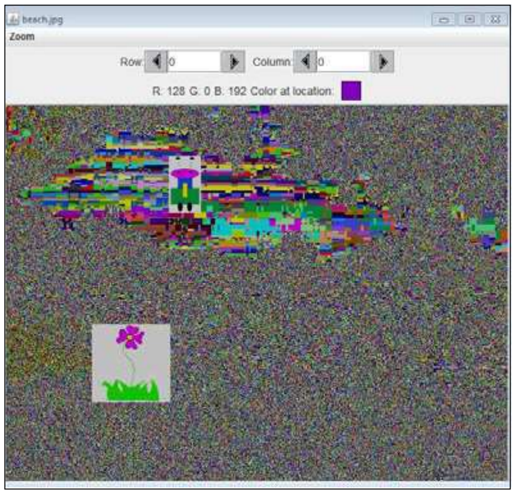
Sample code:

```
Picture beach = new Picture("beach.jpg");
Picture robot = new Picture("robot.jpg");
Picture flower1 = new Picture("flower1.jpg");
beach.explore();

// these lines hide 2 pictures
Picture hidden1 = hidePicture(beach, robot, 65, 208);
Picture hidden2 = hidePicture(hidden1, flower1, 280, 110);
hidden2.explore();

Picture unhidden = revealPicture(hidden2);
unhidden.explore();
```

Results:

`hidden2` ***With Pictures Hidden:***	`unhidden` ***After Calling*** `revealPicture`***:***
 beach.jpg Zoom Row: 0 Column: 0 R: 2 G: 4 B: 3 Color at location:	 beach.jpg Zoom Row: 0 Column: 0 R: 128 G: 0 B: 192 Color at location:

2. Write a method `isSame` that takes two pictures and checks to see if the two pictures are exactly the same (returns `true` or `false`).

Sample code:

```
Picture swan = new Picture("swan.jpg");
Picture swan2 = new Picture("swan.jpg");
System.out.println("Swan and swan2 are the same: " +
   isSame(swan, swan2));
swan = testClearLow(swan);
System.out.println("Swan and swan2 are the same (after clearLow run on swan): "
  + isSame(swan, swan2));
```

Results:

```
Swan and swan2 are the same: true

Swan and swan2 are the same (after clearLow run on swan): false
```

3. Write a method `findDifferences` that takes two pictures and makes a list of the coordinates that are different between them (returns an `ArrayList` of points/coordinates).

Sample code:

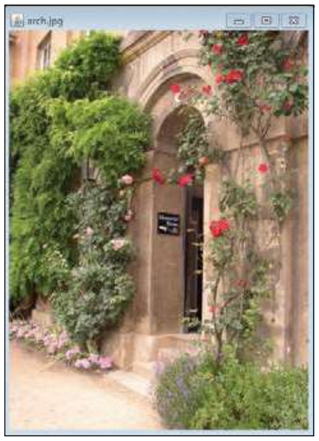
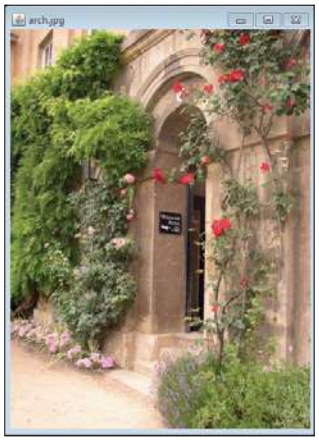
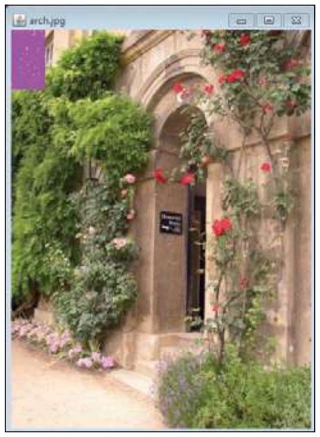
```
Picture arch = new Picture("arch.jpg");
Picture koala = new Picture("koala.jpg");
Picture robot1 = new Picture("robot.jpg");
ArrayList<Point> pointList = findDifferences(arch, arch2);
System.out.println("PointList after comparing two identical pictures " +
   "has a size of " + pointList.size());
pointList = findDifferences(arch, koala);
System.out.println("PointList after comparing two different sized pictures " +
   "has a size of " + pointList.size());
Picture arch2 = hidePicture(arch, robot1, 65, 102);
pointList = findDifferences(arch, arch2);
System.out.println("Pointlist after hiding a picture has a size of  "
   + pointList.size());
arch.show();
arch2.show();
```

Results:

```
PointList after comparing two identical pictures has a size of 0

PointList after comparing two different sized pictures has a size of 0

PointList after hiding a picture has a size of 2909
```

Original Arch	*Arch with Hidden Robot*	*Points in* `PointList` *are Colored*
		

4. Write a method `showDifferentArea` that takes a picture and an `ArrayList` and draws a rectangle around part of picture that is different (returns a `Picture`).

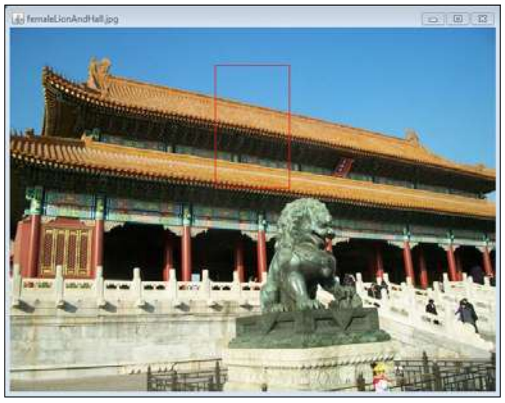
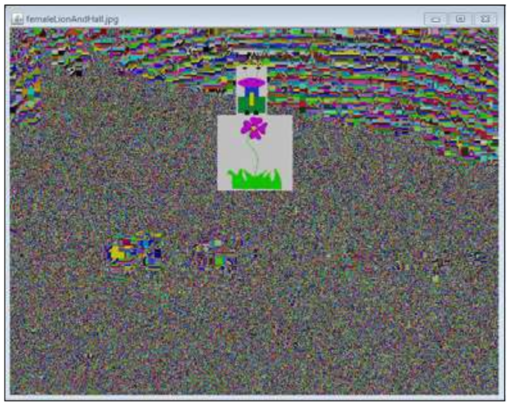
Tip

To provide instructions for the computer to process many different input values, selection statements may need to be nested together to form more than two branches and options. Pathways can be broken down into a series of individual selection statements based on the conditions that need to be checked and nesting together the conditions that should only be checked when other conditions fail or succeed.

Sample code:

```
Picture hall = new Picture("femaleLionAndHall.jpg");
Picture robot2 = new Picture("robot.jpg");
Picture flower2 = new Picture("flower1.jpg");
// hide pictures
Picture hall2 = hidePicture(hall, robot2, 50, 300);
Picture hall3 = hidePicture(hall2, flower2, 115, 275);
hall3.explore();
if(!isSame(hall, hall3))
{
   Picture hall4 = showDifferentArea(hall,
      findDiffferences(hall, hall3));
   hall4.show();
   Picture unhiddenHall3 = revealPicture(hall3);
   unhiddenHall3.show();
}
```

Results:

Bounding Rectangle	*Unhidden Pictures*
	

Check Your Understanding:

5. This activity is focused on being able to hide the secret image in a random location within the source image. One aspect that was not mentioned was how to determine an appropriate random location to hide the secret image. Assuming source and secret are both known (and therefore the height and width of each could be determined through method calls), how would you generate random `row` and `column` values to start hiding secret?

Complete the following expressions:

`int row =` `Math.random() * (source.getHeight() - secret.Height() - 1)`

`int column =` `Math.random() * (source.getWidth() - secret.getWidth() - 1)`

6. Are your coordinates guaranteed to fit secret within source? If not, modify the above expressions to ensure that there is room to fit the entire secret image within source.

Yes, my coordinates are guaranteed to fit!