Questions

1. Open `Picture.java` and look for the method `getPixels2D`. Is it there? No
2. Open `SimplePicture.java` and look for the method `getPixels2D`. Is it there? yes
3. Does the following code compile? No
```
DigitalPicture p = new DigitalPicture();
```
4. Assuming that a no-argument constructor exists for `SimplePicture`, would the following code compile? yes
```
DigitalPicture p = new SimplePicture();
```
5. Assuming that a no-argument constructor exists for `Picture`, does the following code compile? yes
```
DigitalPicture p = new Picture();
```
6. Assuming that a no-argument constructor exists for `Picture`, does the following code compile? yes
```
SimplePicture p = new Picture();
```
7. Assuming that a no-argument constructor exists for `SimplePicture`, does the following code compile? No
```
Picture p = new SimplePicture();
```

`DigitalPicture` is an *interface*. An *interface* most often only has public abstract methods. An *abstract method* is not allowed to have a body. Notice that none of the methods declared in `DigitalPicture` have a body. If a method can't have a body, what good is it?

Interfaces are useful for separating **what** from **how**. An interface specifies **what** an object of that type needs to be able to do but not **how** it does it. You cannot create an object using an interface type. A class can *implement* (*realize*) an interface as `SimplePicture` does. A non-abstract class provides bodies for all the methods declared in the interface, either directly or through inheritance. You can declare a variable to be of an interface type and then set that variable to refer to an object of any class that implements that interface. For example, Java has a `List` interface that declares the methods that a list should have such as `add`, `remove`, and `get`, etc. But, if you want to create a `List` object you will create an `ArrayList` object. It is recommended that you declare a variable to be of type `List`, not `ArrayList`, as shown below (for a list of names).

```
List<String> nameList = new ArrayList<String>();
```

Why wouldn't you just declare `nameList` to be of the type `ArrayList<String>`? There are other classes in Java that implement the `List` interface. By declaring `nameList` to be of the type `List<String>` instead of `ArrayList<String>`, it is easy to change your mind in the future and use another class that implements the same interface. Interfaces give you some flexibility and reduce the number of changes you might need to make in the future, as long as your code only uses the functionality defined by the interface.

Because `DigitalPicture` declares a `getPixels2D` method that returns a two-dimensional array of `Pixel` objects, `SimplePicture` implements that interface, and `Picture` inherits