Part II:

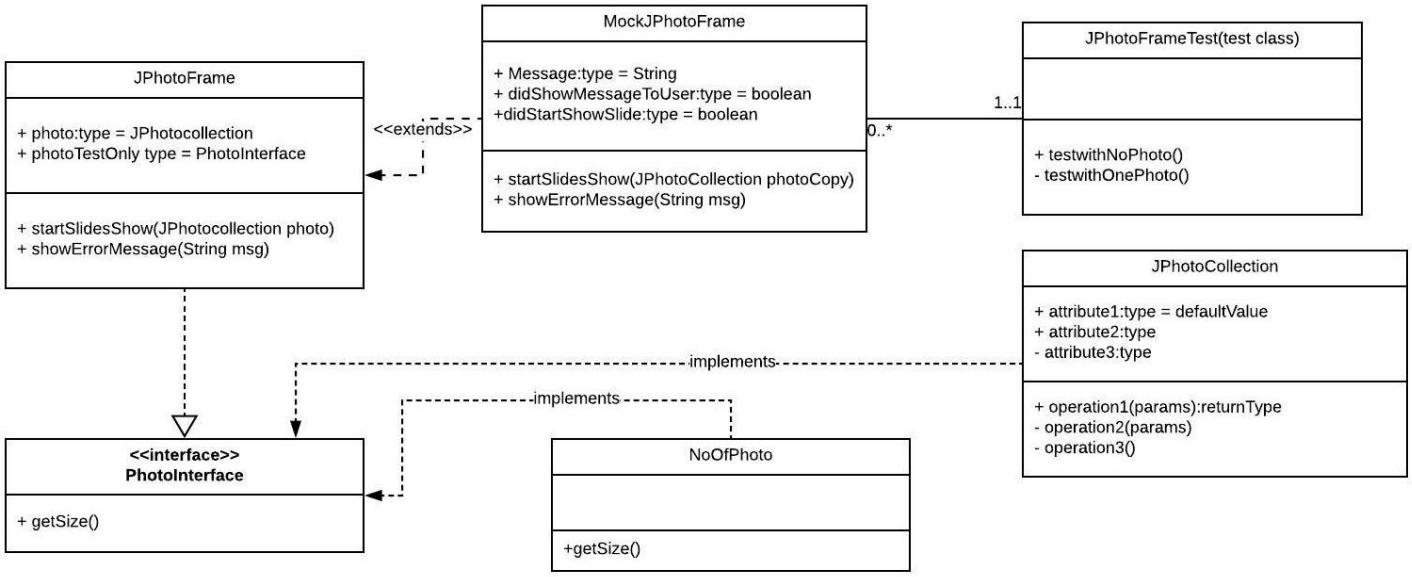

Figure 1: UML

To perform unit test for "Start Slideshow" button, we extract the related code block from `actionPerformed()` method in `JPhotoFrame` class, so that some mock objects can call that part of code without executing other parts. Afterwards, in `JPhotoFrameTest` class we create a subclass `TestingJPhotoFrame` that extends `JPhotoFrame` class, to instantiate some mock objects. The subclass has some overridden methods which has different utility from its super class, i.e., instead of display slides (if size of photo album > 0) or prompt error message (if size of photo album $= 0$), methods in subclass can be overwitten to recording some actual value that help us testing by assserting to some expected value.
We perform two test cases to ensure that wo possibilities after clicking is both correct. In the first test case, we have a abstract frame (Not real Photo frame window) which contains no photos and test whether it will show correct error message with expected `"No photos to show!"` The second one we have another abstract frame which has exactly one abstract photo in album and test whether it will show the slide (it will not show the slide in real, but returns a `boolean` value to pretending it is showing). More logically, instead of using the `MockJPhotoFrame` in the test cases we can break dependency by instanciating `NoOfPhoto`, so that we can get rid of calling other part in `JPhotoCollection`,

```
// testing "Start Slide show" feature
else if (cmd.equals(JPhotoMenu.A_SLIDESHOW)) {
    showPhotoSlides(photos);
}
void showPhotoSlides(JPhotoCollection photoCopy) {
    if (photoCopy.getSize()>0) {
        startSlidesShow(photoCopy);
    }
    else {
        showErrMessage("No photos to show!");
    }
}

public void startSlidesShow(JPhotoCollection photos2) {
    JPhotoShow show = new JPhotoShow(photos2, 5000, list);
    show.setVisible(true);
}

public void showErrMessage(String msg) {
    JOptionPane.showMessageDialog(this, msg,
          APP_NAME, JOptionPane.ERROR_MESSAGE);
}
```

```
class MockJPhotoFrame extends JPhotoFrame {

    public String Message;
    public boolean didShowMessageToUser;
    public boolean didStartShowSlide;

    protected MockJPhotoFrame() throws Exception {
        super();
    }

    @Override
    public void showErrMessage(String msg) {
        didShowMessageToUser = true;
        Message = msg;
    }

    @Override
    public void startSlidesShow(JPhotoCollection photoCopy) {
        this.didStartShowSlide = true;

    }
}
```

```
public class JPhotoFrameTest {
    @Test
    public void testWithNoPhoto() throws Exception {
        MockJPhotoFrame testOne = new MockJPhotoFrame();
        JPhotoCollection empty = new JPhotoCollection();

        testOne.showPhotoSlides(empty);

        assertTrue(testOne.didShowMessageToUser);
        assertEquals("No photos to show!", testOne.Message );
    }

    @Test
    public void testWithOnePhoto() throws Exception {
        MockJPhotoFrame testTwo = new MockJPhotoFrame();
        JPhotoCollection onePhoto = new JPhotoCollection(new String[] { "photo" });
        testTwo.showPhotoSlides(onePhoto);
        assertTrue(testTwo.didStartShowSlide);
    }
}
```

Figure 2: Extracting behaviour out; Subclass for testing; Unit test class conducting 2 test cases

Testing things are all passed in as parameters so that this code is hard to test. In terms of tradeoffs, instead of using PhotoInterface we instantiate the JPhotoFrame and pass some range string to make the photo size > 0 or we can instantiate the JPhotoFrame by using the default constructor. We make this tradeoff because instantiating the JPhotoFrame does not have a lot of external dependencies and is less time-consuming. By using the interface in a large-scale project, we might introduce an anonymous bug into the preious code so that it becomes more time-consuming.
Testing driven design may change the code structure because a common intuition for the developer is to satisfy the testing rather than implementing good structure.

To add new feature as "Quick Slideshow" we add another button in `JPhotoMenu` class, as well as another event (`elseif` block) under `actionPerformed()` method in `JPhotoFrame` class, where we perform similar implementation as original Slideshow apart from changing parameters passed into `JPhotoShow` from `5000` to `50`, which means reducing time intervals from 5000ms to 50ms. Similar unit tests can be applied and details can be found on GitLab:
https://gitlab.doc.ic.ac.uk/jz2618/jphotoalbum.git