1) Write a function that detects if two strings are anagram e.g. ‘bleat’ and ‘table’ are anagrams but ‘eat’ and ‘tar’ are not.

Solution :

```java
package com.shadhin;

import java.util.Scanner;

public class AnagramsFinder {

    public static void main(String[] args) {
        String firstString = "";
        String secondString = "";
        boolean isAnagram = false;

        Scanner Obj = new Scanner(System.in);
        System.out.print("Enter First String : ");
        while (Obj.hasNext()) {
            firstString = Obj.nextLine();
            System.out.print("Enter Second String : ");
            secondString = Obj.nextLine();

            isAnagram = checkAnagram(firstString, secondString);
            if (isAnagram) {
                System.out.println("Strings are Anagram");
            } else {
                System.out.println("Strings are not Anagram");
            }

            System.out.print("Enter First String : ");
        }
    }

    public static boolean checkAnagram(String firstString, String secondString) {
        //If two string match then print Strings are Anagram

        int firstStringLen;
        int secondStringLen;
        //remove space and convert to lowercase from lowercase and uppercase
        firstString = firstString.toLowerCase().replaceAll(" ", "");
        //remove space and convert to lowercase from lowercase and uppercase
        secondString = secondString.toLowerCase().replaceAll(" ", "");

        firstStringLen = firstString.length();
        secondStringLen = secondString.length();
        boolean isAnagramFirst = false;
        boolean isAnagramSecond = false;
        if (firstStringLen > 0 && secondStringLen > 0) {
            if (firstStringLen != secondStringLen) {
                return false;
            } else {
                String[] splitedFirstString = firstString.split("");
                String[] splitedSecondString = secondString.split("");

                for (int i = 0; i < firstStringLen; i++) {
                    isAnagramFirst = isStringContains(splitedFirstString[i],
secondString);
                    isAnagramSecond = isStringContains(splitedSecondString[i],
firstString);

                    if (!isAnagramFirst || !isAnagramSecond) {
                        return false;
                    }
                }
                return true;
            }
        } else {
            System.out.println("Please input valid string");
        }
        return false;
    }

    private static boolean isStringContains(String firstString, String secondString)
{
        //if letter found in world then return true
        return secondString.contains(firstString);
    }
}
```

UNIT Test

```java
package com.shadhin.test;

import com.shadhin.AnagramsFinder;
import org.junit.Assert;

import static org.hamcrest.CoreMatchers.is;

public class AnagramsTest {
    AnagramsFinder SUT;

    @org.junit.Before
    public void setUp() throws Exception {
        SUT = new AnagramsFinder();
    }

    @org.junit.Test
    public void checkAnagram_emptyString_falseWarningReturn() {
        boolean result = AnagramsFinder.checkAnagram("", "");
        Assert.assertThat(result, is(false));
    }

    @org.junit.Test
    public void checkAnagram_lengthNotSame_falseReturn() {
        boolean result = AnagramsFinder.checkAnagram("shadhin", "shadhinshadhin");
        Assert.assertThat(result, is(false));
    }

    @org.junit.Test
    public void checkAnagram_withSpaceValidCheck_trueReturn() {
        boolean result = AnagramsFinder.checkAnagram("shadhin", " s ha dh in");
        Assert.assertThat(result, is(true));
    }

    @org.junit.Test
    public void checkAnagram_withUppercaseLowercaseCheck_trueReturn() {
        boolean result = AnagramsFinder.checkAnagram("SHAdhin", " s ha dh in");
        Assert.assertThat(result, is(true));
    }
    @org.junit.Test
    public void checkAnagram_withLengthSameButNotValidCheck_falseReturn() {
        boolean result = AnagramsFinder.checkAnagram("SHAdhin", " h ha dh in");
        Assert.assertThat(result, is(false));
    }
    @org.junit.Test
    public void checkAnagram_withEmptyStringAndString_falseReturn() {
        boolean result = AnagramsFinder.checkAnagram("", " h ha dh in");
        Assert.assertThat(result, is(false));
    }
}
```

2) Explain the design pattern used in following:
interface Vehicle {
int set_num_of_wheels()
int set_num_of_passengers()
boolean has_gas()
}
a) Explain how you can use the pattern to create car and plane class?

b) Use a different design pattern for this solution.

My Solution:

2) This design pattern is called **Abstract factory** design pattern.

a) This Abstract factory is created using interfaces. The same can be done by simply replacing the interface with an abstract class and instead of implementing the interface, the sub-classes will extend the abstract class and the methods declared in the abstract class must be implemented in each of the sub-classes. The factory and the main method stay the same in both cases. Here I will implements the Vehicle interface in Car and Plane Class and the methods declared in the interface Vehicle must be implemented in each of the classes (Car and Plane). Example :

```
public class Car implements Vehicle {
    @Override
    public int set_num_of_wheels() {
        return 4;
    }

    @Override
    public int set_num_of_passengers() {
        return 4;
    }

    @Override
    public boolean has_gas() {
        return true;
    }
}
```

```java
public class Plane implements Vehicle{

    public static void main(String[] args) {
   // write your code here
    }

    @Override
    public int set_num_of_wheels() {
        return 4;
    }

    @Override
    public int set_num_of_passengers() {
        return 100;
    }

    @Override
    public boolean has_gas() {
        return false;
    }
}
```

b) **Factory design pattern** : Design patterns are often used to simplify big chunks of code or even to hide specific implementations from the application flow. There are lots of design patters in java, the perfect example for these kind of problems is the **factory design pattern** and it is very easy to understand.

It proposes using a super class and multiple sub-classes, which inherit from the super class. During execution only the super class is used and its value varies depending on the factory class. In order to use this pattern, we need to implement a factory class, which will return the correct sub-class for a given input. The java classes bellow specify one super class (Vehicle.java) and two sub-classes (Car.java and Plane.java).

```java
//Supper Class
public class Vehicle {

    int set_num_of_passengers() {
        return 100;
    }

    boolean has_gas() {
        return true;
    }

    int set_num_of_wheels() {
        return 4;
    }
}
```

```java
//Sub Class
public class Car extends Vehicle{

    @Override
    int set_num_of_passengers() {
        return 4;
    }

    @Override
    boolean has_gas() {
        return true;
    }
}
```

```java
//Sub class
public class Plane extends Vehicle {
    @Override
    boolean has_gas() {
        return false;
    }
}
```

UNIT TEST

```java
package com.shadhin.unitest;

import com.shadhin.my_solution_factory_design_pattern.Car;
import org.junit.Assert;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.*;

public class CarTest {
    Car SUT;

    @org.junit.Before
    public void setUp() throws Exception {
        SUT = new Car();
    }

    @org.junit.Test
    public void checkCar_gasNeed_trueReturn() {
        boolean result = SUT.has_gas();
        Assert.assertThat(result, is(true));
    }

    @org.junit.Test
    public void checkCarPassenger_trueReturn() {
        int result = SUT.set_num_of_passengers();
        Assert.assertTrue(100 > result);
    }

    @org.junit.Test
    public void checkCarwheels4_trueReturn() {
        int result = SUT.set_num_of_wheels();
        Assert.assertEquals(4, result);
    }

    @org.junit.After
    public void tearDown() throws Exception {
    }
}
```

```java
package com.shadhin.unitest;

import com.shadhin.my_solution_factory_design_pattern.Plane;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.*;

public class PlaneTest {
    Plane SUT;

    @Before
    public void setUp() throws Exception {
        SUT = new Plane();
    }

    @org.junit.Test
    public void checkPlan_gasNeed_falseReturn() {
        boolean result = SUT.has_gas();
        Assert.assertThat(result, is(false));
    }

    @org.junit.Test
    public void checkPlanPassenger_trueReturn() {
        int result = SUT.set_num_of_passengers();
        Assert.assertTrue(result > 99);
    }

    @org.junit.Test
    public void checkPlanwheels4_trueReturn() {
        int result = SUT.set_num_of_wheels();
        Assert.assertEquals(4, result);
    }

    @After
    public void tearDown() throws Exception {
    }
}
```

3) Write a video player application with ‘Play’, ‘Forward’ , ‘Rewind’ functionalities. Please write pseudocode for this program and explain the design pattern you will use to develop these three functionalities.

Solution :

Pseudocode for this program

```
public void playVideo(View v){
        MediaController m=new MediaController(this);
        video.setMediaController(m);
        String path="android/"+R.raw.trial;
        Uri u=Uri.parse(path);
        video.setVideoURI(u);
        video.start();
        }
```

```
public void playForward(View v){
        int time=(int)startTime;

        if((time+forward_Time)<=final_time){
        startTime=startTime+forward_Time;
        Player.seekTo((int)startTime);
        Toast.makeText(getApplicationContext(),"Forward for 5
seconds",Toast.LENGTH_SHORT).show();
        }else{
        Toast.makeText(getApplicationContext(),"Cannot
forward",Toast.LENGTH_SHORT).show();
        }
        }
```

```
public void playBack(View v){
        Player.start();
        final_Time=Player.getDuration();
        startTime=Player.getCurrentPosition();

        if(oneTime==0){
        seek.setMax((int)final_Time);
        oneTime=1;
        }
        seek.setProgress((int)startTime);
        handeler.postDelayed(UpdateSongTime,100);
        }
```

Explain the design pattern I will use to develop these three functionalities.

Solution: There are 3 most popular architectural pattern in android development.

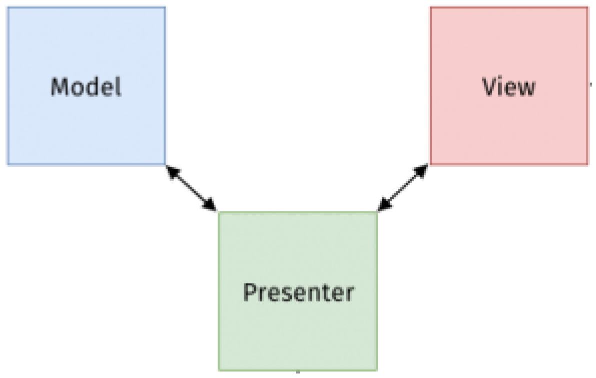
1. MVC
2. MVP
3. MVVM

For in this application I will use MVP architectural pattern.
MVP is an architecture pattern that you can use to deal with some of the shortcomings of MVC, and is a good alternative architecture. It provides an easy way to think about the structure of your app. It provides modularity and testability. To address the architectural issues, MVP separates the application into three layers where all user events are delegated to Presenter.

Model : Model is responsible for managing the data, irrespective of the source of data. The responsibilities of model are to cache data and validate the data source; either network or local data source like sqlite, files etc. A model is often implemented via Repository Pattern to abstract away the internal implementation of data source.

View : The only job of view is to display the content instructed by the presenter. The View interface can be implemented by Activity, Fragment, or it's subtypes. View keeps a reference to the presenter to delegate the user event to the presenter and perform UI oriented tasks like populating lists, showing dialogs, or updating content on screen.

Presenter : The Presenter is an intermediate layer which bridges the communication gap between Model and View. As per the user event, passed by the view, the Presenter queries the model, transforms the data, and passes the updated data to View to be displayed on the screen.



Here I will show you how I design my app using the **MVP** design pattern with **UnitTest**

This is My Presenter Class :

```
package com.era.bongo;

public class VideoPlayerPresenter {
    private View view;

    public VideoPlayerPresenter(View view) {
        this.view = view;
        boolean play = playVideoPresenter();
        boolean forward = forwardVideoPresenter();
        boolean back = playBackVideoPresenter();
    }

    public VideoPlayerPresenter() {
    }

    public boolean playVideoPresenter() {
        String playing = "playing video";
        view.playVideo(playing);
        return true;
    }

    public boolean forwardVideoPresenter() {
        String forward = "forward video";
        view.playForward(forward);
        return true;
    }

    public boolean playBackVideoPresenter() {
        String playBack = "playBack video";
        view.playBack(playBack);
        return true;
    }

    public interface View {
        void playVideo(String playing);

        void playForward(String forward);

        void playBack(String Backward);
    }
}
```

This is my View Class

```java
package com.era.bongo;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.Toast;

public class VideoPlayerViewActivity extends AppCompatActivity implements
VideoPlayerPresenter.View {
    private VideoPlayerPresenter presenter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_video_player_view);
        presenter = new VideoPlayerPresenter(this);
    }

    @Override
    public void playVideo(String playing) {
        Toast.makeText(this, playing, Toast.LENGTH_SHORT).show();
    }

    @Override
    public void playForward(String forward) {
        Toast.makeText(this, forward, Toast.LENGTH_SHORT).show();
    }

    @Override
    public void playBack(String Backward) {
        Toast.makeText(this, Backward, Toast.LENGTH_SHORT).show();
    }


}
```

This is my Model Class

```java
package com.era.bongo;

public class VideoPlayerModel {
}
```

and Finally the unit testing

```java
package com.era.bongo;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import static org.hamcrest.CoreMatchers.is;

public class VideoPlayerPresenterTest {
    @Mock
    VideoPlayerViewActivity mvpView;
    VideoPlayerPresenter SUT;

    @Before
    public void setUp() throws Exception {
        MockitoAnnotations.initMocks(this);
        SUT = new VideoPlayerPresenter(mvpView);
    }

    @org.junit.Test
    public void checkVideoPlayer_Playing_trueReturn() {
        boolean result = SUT.playVideoPresenter();
        Assert.assertThat(result, is(true));
    }

    @org.junit.Test
    public void checkVideoPlayer_forward_trueReturn() {
        boolean result = SUT.forwardVideoPresenter();
        Assert.assertThat(result, is(true));
    }

    @org.junit.Test
    public void checkVideoPlayer_playBack_trueReturn() {
        boolean result = SUT.playBackVideoPresenter();
        Assert.assertThat(result, is(true));
    }

    @After
    public void tearDown() throws Exception {
    }
}
```

Thank You.