Ans to the question1:

```
//As it has only one channel which is named "Twitter", no other channel can be made
//Notify its users once a video is released
// so we will use singleton and observer patterns
// Singleton pattern  help to create "Twitter" channel. that is the only instance
// Observer pattern help  to create a subscribe-notify system

import java.util.ArrayList ;
import java.util.List ;
interface Youtube_channel  {  //initially subscriber,unscriber,video name
  void subscribe(subscriber_user subscriber) ;
  void notification(String video)  ;
  void remove_subscribe(subscriber_user subscriber) ;
}
class channel_twitter implements Youtube_channel  {
  private static channel_twitter channel ;
  private static String name ;
  private List<subscriber_user> subscribers = new ArrayList<>() ; //creates a array
  private channel_twitter() {
  }

  public static channel_twitter get_the_channel() {

    if (channel == null ) {
      synchronized (channel_twitter.class) {

        if (channel == null ) {
          channel = new channel_twitter() ;
          channel.name = "twitter" ;
        }
      }
    }

    return channel;
  }

  public String get_name() {
    return name;
  }
  @Override   //removing subscriber
  public void remove_subscribe( subscriber_user subscriber)  {
    subscribers.remove(subscriber) ;
    System.out.println( subscriber.getUserName()  + " successfully unsubscribed  " + this.name + " youtube Channel."
) ;
  }
  @Override
  public void subscribe( subscriber_user subscriber) {  //adding subscriber
    subscribers.add(subscriber) ;
    System.out.println( subscriber.getUserName() + " successfully subscribed " + this.name  +  " youtube Channel." ) ;
  }
  @Override
```

```
  public void notification(String video) {  // notification
    for (subscriber_user subscriber : subscribers) {
      subscriber.change( video) ;
    }
  }
}   //////

interface Subscriber {
void change(String video) ;
}
class subscriber_user implements Subscriber {
  private String name ;
  public subscriber_user(String name) {
    this.name=name ;
  }
  @Override
  public void change(String video) {

    System.out.println(this.name + " got a notification from " + channel_twitter.get_the_channel().get_name() + "
Youtube Channel. \"" + video  + "\"  uploaded just now");
  }
  public String getUserName() {

    return name ;
  }
}
//////////////////////////
public class Main {

  public static void main(String[] args) {
    channel_twitter channel = channel_twitter.get_the_channel() ;
    subscriber_user people1 = new subscriber_user("Nowshin") ;
    subscriber_user  people2 = new subscriber_user("Sumaiya") ;
    subscriber_user  people3 = new subscriber_user("yen") ;
    channel.subscribe( people1) ; // 3 user subscribe
    channel.subscribe( people2) ;
    channel.subscribe( people3) ;
    channel.notification("twitter 1st video") ;  //1st video
    channel.remove_subscribe(people3); // user 3 unscribe
    channel.notification("twitter 2nd video ") ; //2nd video
  }
}
```

Ans to the question2:

```
//As it has multiple  channels
//Notify its users once a video is released
// so we will use Partial Singleton (Eager initialization) and observer patterns
//partial Singleton for managing Twitter channel, other channels will created normally
// Observer pattern help  to create a subscribe-notify system
import java.util.ArrayList ; //for array, list
import java.util.List ;
interface Youtube_channel {
  void subscribe(subscriber_user subscriber) ;
  void notification(String video) ;
  void remove_subscribe(subscriber_user subscriber) ;
}
class Youtube_channel_name implements Youtube_channel {
  private static Youtube_channel_name channel_twitter = new Youtube_channel_name("twitter") ;
  private String temp ;  //for channel name
  private List<subscriber_user> subscribers=new ArrayList<>() ;
  private Youtube_channel_name(String a) {
    this.temp = a ;
  }

  public static Youtube_channel_name get_the_channel(String a) {
    if (a.equals("twitter"))  {  //only for twitter channel
      return channel_twitter ;

    }
    return new Youtube_channel_name(a) ;
  }
  public String get_the_name() {
    return temp;
  }
  @Override
  public void remove_subscribe(subscriber_user subscriber) {

    subscribers.remove(subscriber);
    System.out.println(subscriber.get_the_user_name() + "  successfully unsubscribed " + this.temp + " Youtube
Channel.");
  }
  @Override
  public void subscribe(subscriber_user subscriber) {

    subscribers.add(subscriber);
    System.out.println(subscriber.get_the_user_name() + "  successfully subscribed " + this.temp + " Youtube
Channel.");
  }
  @Override
  public void notification (String video) {
```

```
    for (subscriber_user subscriber : subscribers) {
      subscriber.update(this.temp, video) ;
    }
  }
}
interface Subscriber {
  void update(String temp,String video) ;
}
class subscriber_user implements Subscriber {
  private String a;
  public subscriber_user(String a) {

    this.a = a;
  }
  @Override
  public void update(String temp, String video) {
    System.out.println(this.a + " got a notification from " + temp + " Youtube Channel. \"" + video + "\" uploaded just
now");
  }
  public String get_the_user_name() {

    return a;
  }
}
///////
public class Main {
  public static void main(String[] args ) {

    Youtube_channel_name channel = Youtube_channel_name.get_the_channel("Linus tech tips");
    Youtube_channel_name only_twitter_channel = Youtube_channel_name.get_the_channel("twitter");
    subscriber_user people1 = new subscriber_user("Nowshin");  //3 user
    subscriber_user people2 = new subscriber_user("Sumaiya");
    subscriber_user people3 = new subscriber_user("yen");

    channel.subscribe(people1); //other subscribe
    channel.subscribe(people2);
    channel.subscribe(people3);
    channel.notification(" 1st video");

    only_twitter_channel.subscribe(people1); // twitter subscribe
    only_twitter_channel.subscribe(people2);
    only_twitter_channel.subscribe(people3);
    only_twitter_channel.notification("Twitter 1st video");

    channel.remove_subscribe(people1);  // remove subscribe
    channel.notification(" 2nd video");

  }
}
```

Ans to the question3:

```
//there will be only one application that can solve the problem of all the mentioned Applications
//so we will use singleton design pattern
// this  ensure that there is only one instance of the wechat application running

interface we_chat_app {
    void video() ;
    void message() ;
    void news_feed() ;
    void friends_group() ;

 }

  class we_chat implements we_chat_app {
    private static we_chat a ;
    private we_chat () {
    }
    public static we_chat getInstance(){

        if (a == null) {
        synchronized (we_chat.class) {

          if (a == null) {
            a = new we_chat();
          }
        }
      }

       return a  ;
    }
    @Override
    public void video(){  //streaming option like Twitch
      System.out.println("you are streaming video.");
    }
    @Override
    public void news_feed(){  //  news feed option like Facebook
      System.out.println("you are using news feed.");
    }
    @Override
    public void message(){  //chatting option like Messenge
      System.out.println("you are sending message.");
    }
    @Override
    public void friends_group() {  //creating friends group  like Facebook
      System.out.println("you are creating friends group.");
    }

    @Override
```

```
  public String toString() {
    return "we chat app is ready with features enabled.";
  }
 }
 // main class
 public class Main {
  public static void main(String[] args ) {
    we_chat we_chat_app = we_chat.getInstance() ;
    we_chat we_chat_app2 = we_chat.getInstance() ;

    we_chat_app.news_feed() ;
    we_chat_app.message() ;
    we_chat_app2.video() ;
    we_chat_app2.friends_group() ;

    System.out.println(we_chat_app) ;
    System.out.println(we_chat_app2) ;
  }
 }
```

Ans to the question4:

```
//their success comes from having one source
//every cake is consistently perfect,just like the first
// so we will use singleton
//singleton pattern ensures that only one instance of a class is created

class magic_sweet_management {
  private static magic_sweet_management baker;
  private magic_sweet_management() {
  }
  public static magic_sweet_management getbaker(){

    if (baker == null) {
      synchronized (magic_sweet_management.class) {

        if (baker == null ) {
            baker = new magic_sweet_management() ;
        }
      }
    }
    return baker;
  }

  public void cooking_sweet() {

    System.out.println("Yay! the magic sweets are being cooked just right!");
  }
  public void sell_sweets() {

    System.out.println("Magic sweets are being sold.");
```

```
  }
  @Override
  public String toString() {
    return "MagicSweetsManager: the one and only instance for magic sweets.";
  }
}
public class Main {
  public static void main(String[] args)  {
    magic_sweet_management people1 = magic_sweet_management.getbaker() ;
    people1.cooking_sweet() ;
    people1.sell_sweets();

    magic_sweet_management people2 = magic_sweet_management.getbaker() ;
    people2.cooking_sweet() ;
    people2.sell_sweets();

    magic_sweet_management people3 = magic_sweet_management.getbaker() ;
    people3.cooking_sweet() ;
    people3.sell_sweets();

    System.out.println(people1) ;
    System.out.println(people2) ;
    System.out.println(people3) ;

  }
}
```