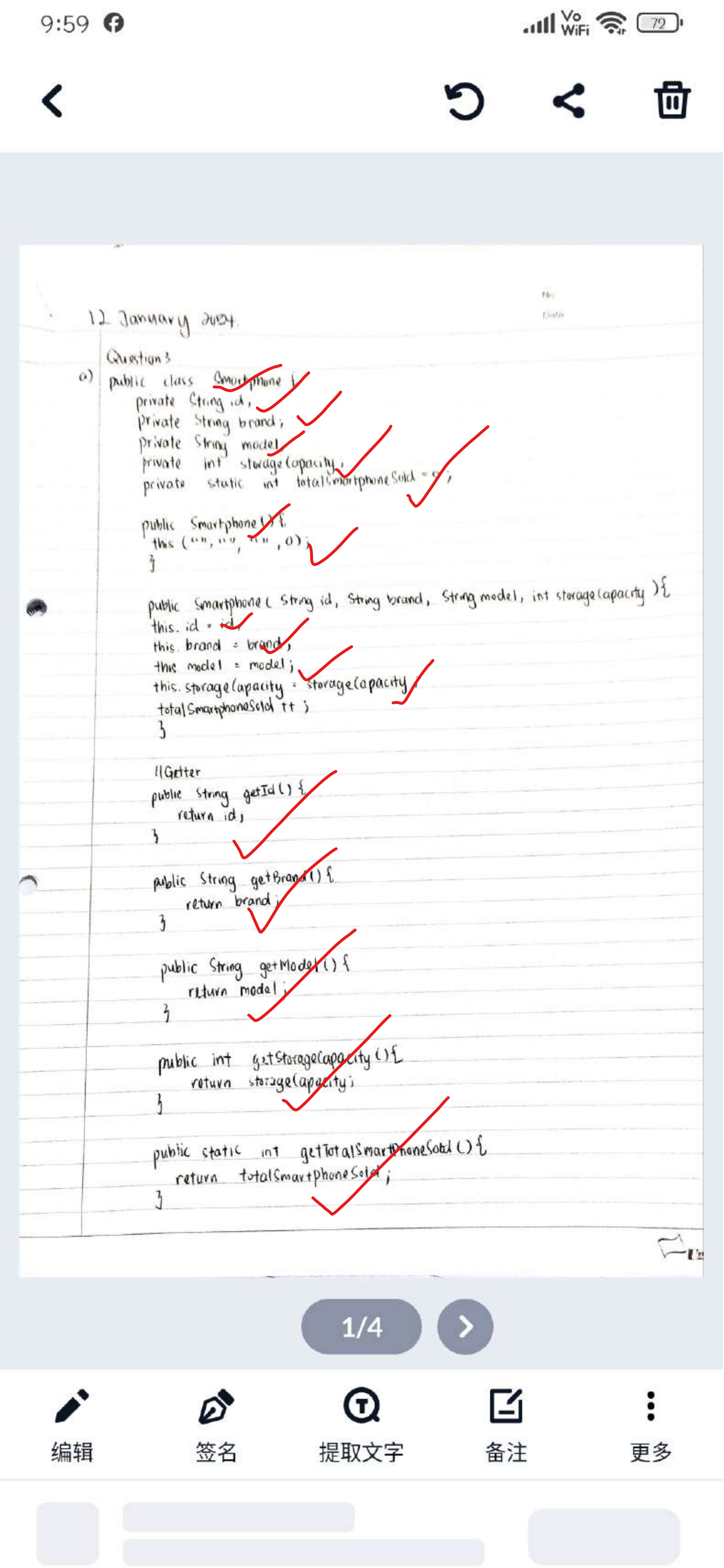
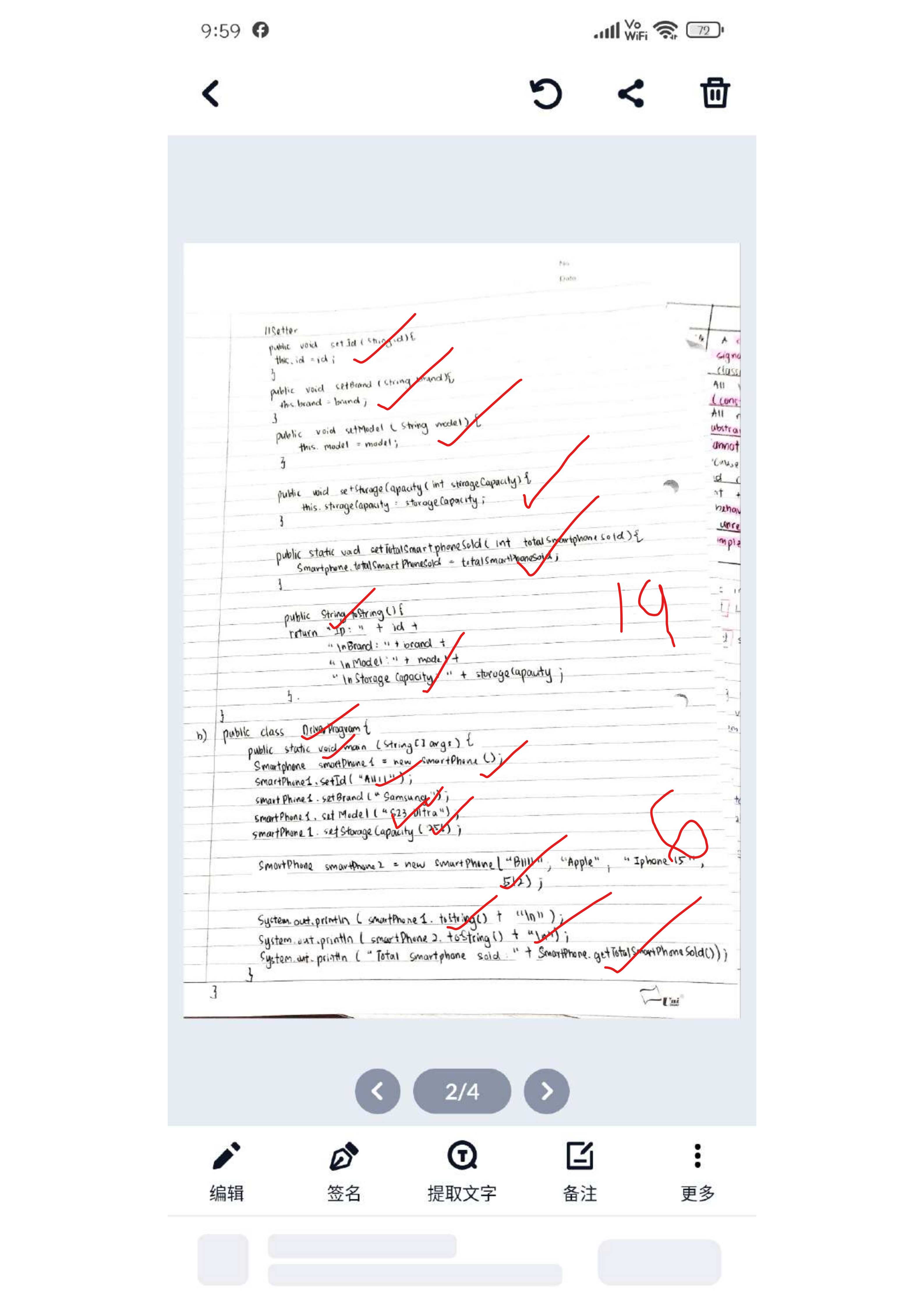


No.
Date

12 January 2024.

Question 3

a)
```
public class Smartphone {
    private String id;
    private String brand;
    private String model;
    private int storageCapacity;
    private static int totalSmartphoneSold = 0;

    public Smartphone(){
        this ("", "", "", 0);
    }

    public Smartphone( String id, String brand, String model, int storageCapacity ){
        this. id = id;
        this. brand = brand;
        this model = model;
        this. storageCapacity = storageCapacity;
        totalSmartphoneSold ++ ;
    }

    //Getter
    public String getId() {
        return id;
    }

    public String getBrand() {
        return brand;
    }

    public String getModel() {
        return model;
    }

    public int getStorageCapacity (){
        return storageCapacity;
    }

    public static int getTotalSmartPhoneSold (){
        return totalSmartphoneSold;
    }
```

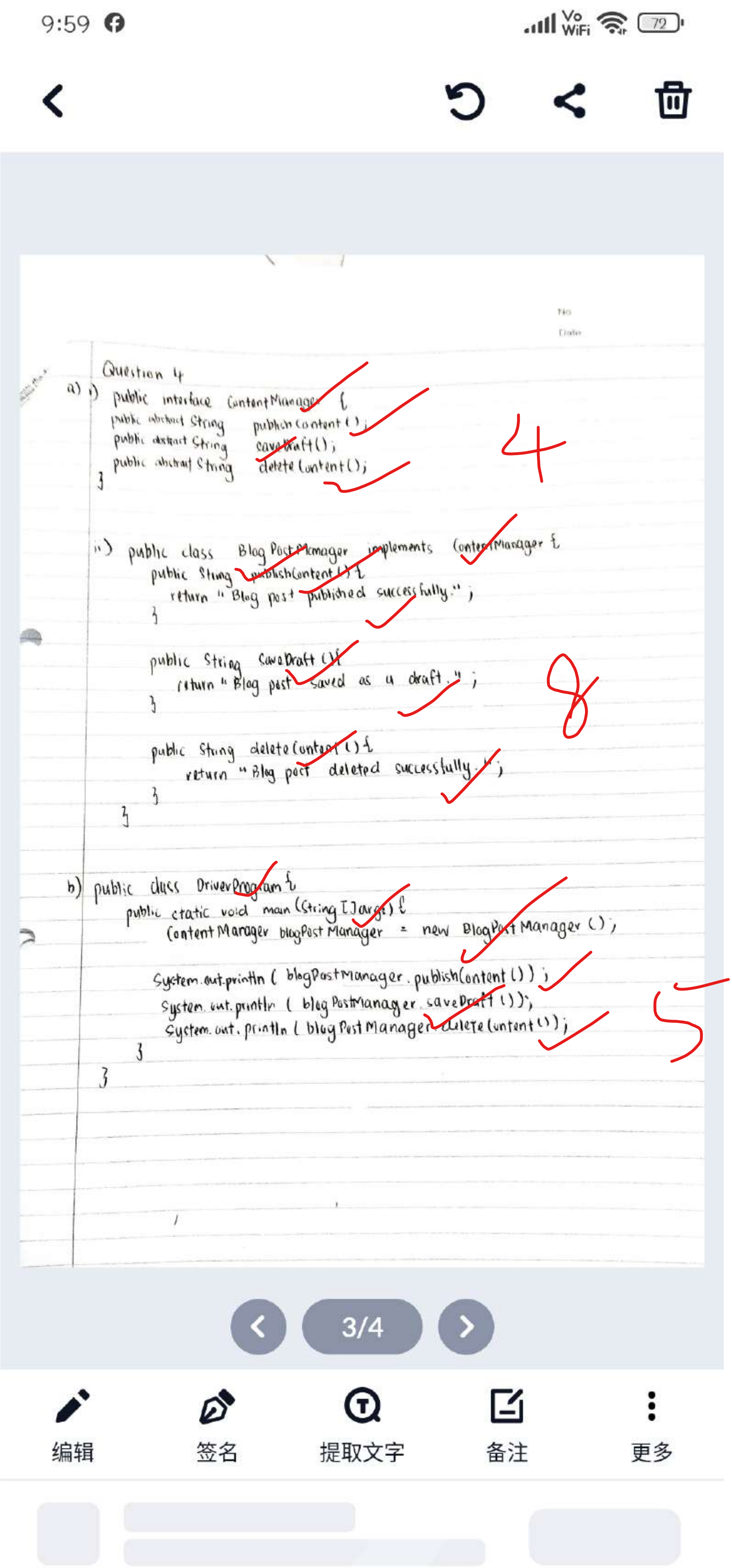


```
//Setter
public void setId (String id){
 this.id = id;
}
public void setBrand (String brand){,
 this.brand = brand;
}
public void setModel (String model){
    this. model = model;
 }

public void setStorageCapacity (int storageCapacity){
    this. storageCapacity = storageCapacity;
}

public static vaid setTotalSmartphoneSold (int totalSmartphoneSold){
   Smartphone.totalSmartPhoneSold = totalSmartPhoneSold;
}

 public String toString(){
 return "ID: " + id +
        "\nBrand: " + brand +
        "\nModel: " + model +
        "\nStorage Capacity: " + storageCapacity;
 }.
}
```

19

```
b) public class DriverProgram {
     public static void main (String[] args) {
      Smartphone smartPhone1 = new SmartPhone ();
      SmartPhone1.setId ("A111");
      smartPhine1.setBrand ("Samsung");
      smartPhone1. set Model ("S23 Ultra");
      smartPhone1. setStorageCapacity (256);

      SmartPhone smartPhone2 = new SmartPhone ("B111", "Apple", "Iphone 15",
                                               512);

      System.out.println (smartPhone1. toString() + "\n");
      System.out.println (smartPhone2. toString() + "\n");
      System.out.println ("Total smartphone sold: " + SmartPhone.getTotalSmartPhoneSold());
     }
}
```

6



No
Date

Question 4

a) i)

```
public interface ContentManager {
public abstract String    publishContent ();
public abstract String    saveDraft();
public abstract String    deleteContent();
}
```

4

ii)

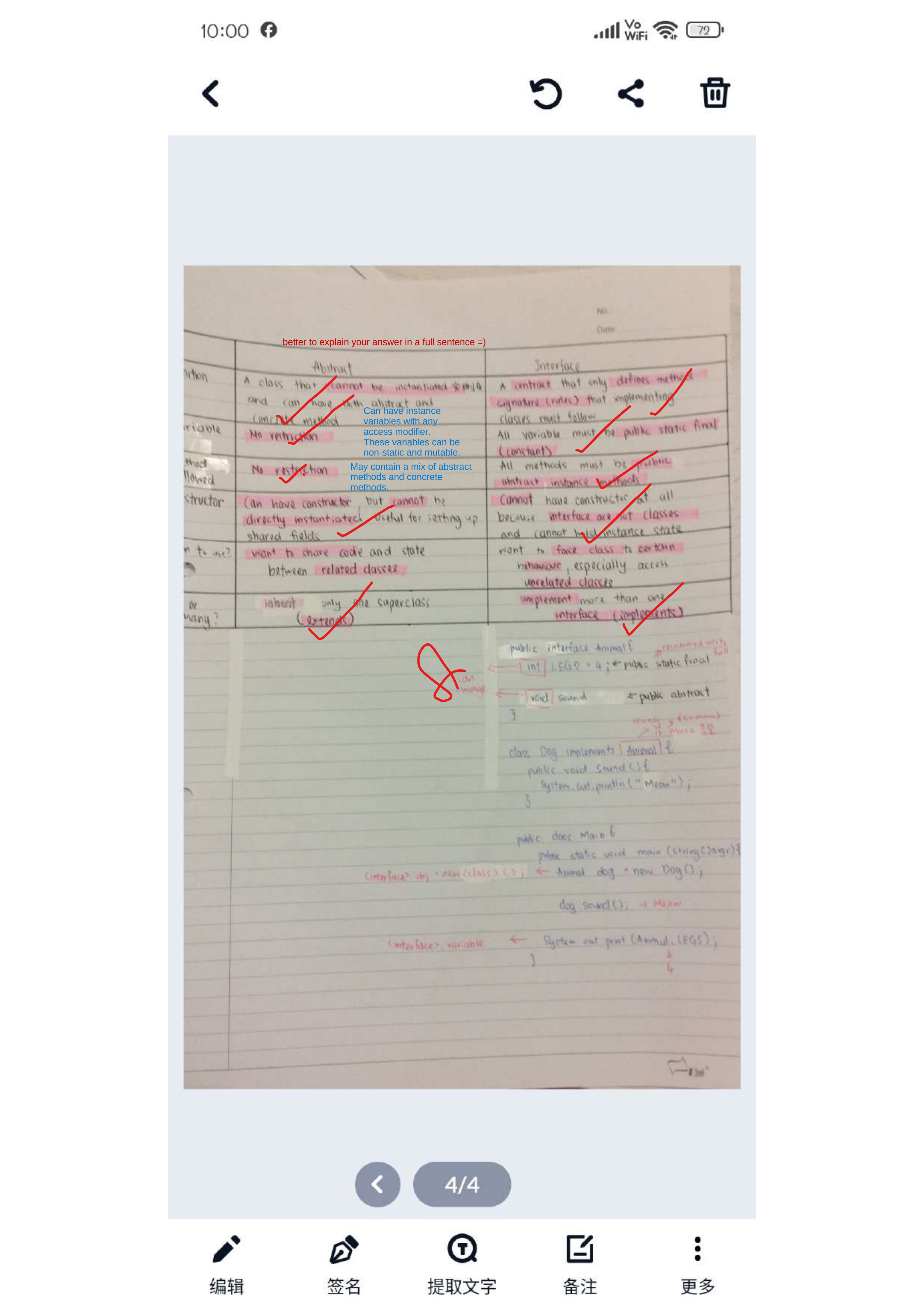
```
public class BlogPostManager implements ContentManager {
    public String publishContent () {
        return " Blog post published successfully." ;
    }

    public String saveDraft (){
        return " Blog post saved as a draft." ;
    }

    public String deleteContent () {
        return " Blog post deleted successfully." ;
    }
}
```

8

b)

```
public class DriverProgram {
    public static void main (String [] args) {
        ContentManager blogPostManager = new BlogPost Manager () ;

        System.out.println ( blogPostManager . publishContent () ) ;
        System.out.println ( blog PostManager . saveDraft ());
        System.out.println ( blog PostManager.deleteContent ());
    }
}
```

5



No.

Date

better to explain your answer in a full sentence =)

	Abstract	Interface
...rtion	A class that cannot be instantiated 实例化 and can have both abstract and concrete method	A contract that only defines methods signature (rules) that implementing classes must follow
...riable	No retriction Can have instance variables with any access modifier. These variables can be non-static and mutable.	All variable must be public static final (constant)
...thod ...llowed	No restriction May contain a mix of abstract methods and concrete methods.	All methods must be public abstract instance methods
...structor	Can have constructor, but cannot be directly instantiated. Useful for setting up shared fields	Cannot have constructor at all because interface are not classes and cannot hold instance state
...n to use?	Want to share code and state between related classes	want to force class to certain behaviour, especially access unrelated classes
... or ...many?	inherit only one superclass (extends)	implement more than one interface (implements)

```
public interface Animal{
    int LEGS = 4; ← public static final
    void sound  ← public abstract
}

class Dog implements Animal {
    public void sound(){
        System.out.println("Meow");
    }
}

public class Main {
    public static void main (String[]agr){
        Animal dog = new Dog();

        dog.sound(); → Meow

        System.out.print (Animal.LEGS);
                                    ↓
                                    4
    }
```

can change

recommend write full

string → (...) more 建议

<interface> obj = new <class> ();

<interface>.variable