Q1

Q1 chỉ bao gồm tạo class, tạo attribute (biến), constructor, getter và setter, override toString(), trong đó phải điều chỉnh các format của getter

1. **uppercase (viết hoa) toàn bộ**

s = s.toUpperCase();

1. **lowercase (viết thường) toàn bộ**

s = s.toLowerCase();

1. **viết thường 3 chữ cái đầu tiên, các chữ còn lại giữ nguyên**

public String getPlace() {

return place.substring(0, 3).toLowerCase() + place.substring(3, place.length());

}

1. viết hoa chữ cái đầu, các chữ còn lại giữ nguyên

public String getPlace() {

return place.substring(0, 1).toUpperCase()+ place.substring(1, place.length());

}

Lưu ý: - cách dùng các method của String là phải dùng phép gán

Q2

Q2 sẽ phải tạo 2 class cha và con, class con được extends từ class cha.

Cả 2 class đều có các attribute, constructor, getter setter, toString (như câu 1).

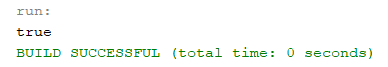
Ngoài ra, class con có thêm 2 method setData() và getData()

# Các hàm, method

1. **Kiểm tra xem 1 ký tự là chữ thường không**



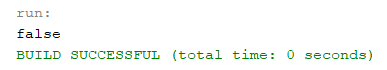
Output:



1. **Kiểm tra xem 1 ký tự là chữ hoa không**



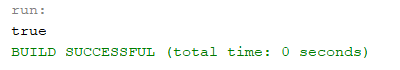
Output:



1. **Kiểm tra 1 ký tự là chữ số hay không**



Output:



1. **Kiểm tra xem 1 ký tự là chữ hay không**

Character.isLetter();

1. **Kiểm tra xem 1 ký tự là chữ hoặc số hay không (từ a-z, A-Z, 0-9)**



output: true

1. **lấy ra 1 chuỗi gồm 5 ký tự đầu của chuỗi str**

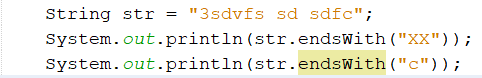
method substring(beginIndex, endIndex)



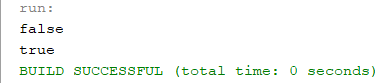
Output:



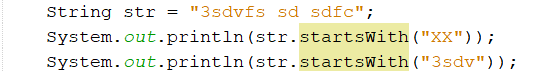
1. **check xem str có kết thúc bằng chuỗi “XX” không (ví dụ ở câu 8 phía dưới)**



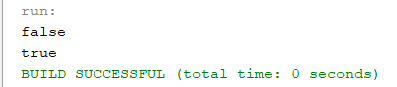
Output:



1. **check xem str có bắt đầu bằng chuỗi “XX” không**



Output:



# Các dạng bài Q2 (2 dạng setData() và getData())

1. **thay thế 2 chữ cái cuối cùng của tailor thành chuỗi “XYZ”**

setData() replace the last 2 characters in *tailor* with the string “XYZ”

public void setData() {

String s = "";

s+=getTailor().substring(0, getTailor().length()-2);

//sao chep (n-2) ki tu dau tien cua tailor vao s

s += "XYZ"; //them "XYZ" vao cuoi s

setTailor(s); //set lai gia tri moi cho tailor

}

ngắn gọn hơn nữa

public void setData() {

setTailor(getTailor().substring(0, getTailor().length()-2) + "XYZ");

}

1. **setData() add string “XZ” to the head of *maker* and increase *price* by 20**

public void setData() {

setMaker(“XZ” + getMaker());

setPrice(getPrice() + 20);

}

1. **setData(): replace the 3rd character in *owner* with the string “XX” (thay thế ký tự thứ 3 (index = 2) của *owner* thành chuỗi “XX”)**

gh3fhhh

ghXXfhhh

public void setData(){

int n = getOwner().length();

setOwner(getOwner().substring(0,2) + "XX" + getOwner().substring(3,n));

}

1. **setData(): void - Add the string “BETA” to the beginning of *maker***

public void setData(){

setMaker(“BETA”+getMaker());

}

1. **getValue() return *price*+inc, where if *type*<7 then inc=10, otherwise inc=15**

public int getValue() {

if (type<7) {

return getPrice() + 10;

} //else

return getPrice + 15;

}

1. **getValue(). check if *tailor* starts with digit the return color+11, otherwise return color+4**

public int getValue() {

if (Character.isDigit(getTailor().charAt(0))) {

return getColor() + 11;

}

return getColor() + 4;

}

1. **getValue():int - Check if *color* is odd number (số lẻ) then return *price*+7, otherwise return *price*+3**

public int getValue(){

if(color % 2 == 1){

return getPrice() +7;

}

else return getPrice() + 3;

}

1. **getValue():int - Check if *maker* ends with “XX” và starts with a digit then return *odor*+15, otherwise return *odor*+10 (*maker* với *odor* đều là attribute của class cha)**

public int getValue(){

if (getMaker.endsWith(“XX”) && Character.isDidit(getMaker.charAt(0))){ //startsWith

return getOdor()+15;

}

return getOdor() + 10;

}

Q3

Viết class theo yêu cầu đề bài (giống Q1)

Đề đã có sẵn 1 cái interface A, nhiệm vụ của mình là tạo một class B implements từ interface A, sau đó thêm vào các hàm theo yêu cầu đề bài (thường là f1, f2, f3)

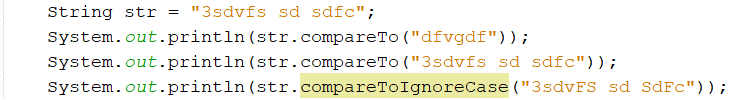


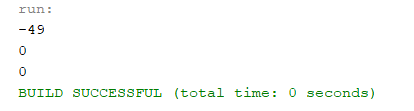
# Các hàm, method

ở Q3 này thường xử lý với ArrayList với String, nên phải biết một số các method cần thiết của ArrayList với String

## String

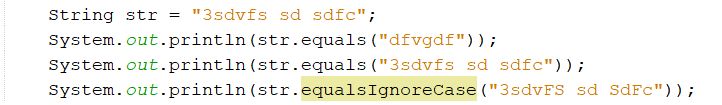
1. **so sánh 2 chuỗi: compareTo, compareToIgnoreCase, equals, equalsIgnoreCase**

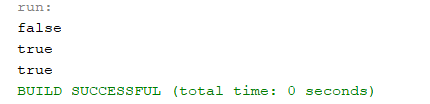




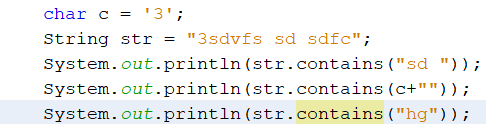
compareTo với compareToIgnoreCase trả về giá trị dương nếu str1 > str2, 0 nếu str1=str2, giá trị âm nếu str1<str2 (so sánh dựa trên chữ cái đầu tiên, => thứ 2, …)

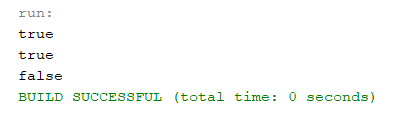
Tương tự với equals, nhưng compare trả về giá trị số nguyên thì equals trả về giá trị boolean (true, false)





1. **contains trả về giá trị boolean (true, false), lưu ý: tham số truyền vào phải là string**





1. **replace //ko hay dùng**
2. **indexOf, lastIndexOf**

## ArrayList

1. contains //đừng dùng, remove//cần thiết cho f2, indexOf
2. subList
3. Collections.sort()
4. Collections.min(), Collections.max()

# Các dạng bài Q3

## F1: tính số/tổng/trung bình phần tử của list thỏa mãn điều kiện đề bài

1. **count and return number of elements having color > average color (the average may be real number, in the sample data below the average is 6)**

@Override

public int f1(List<Cape> t) {

int count = 0;

float avercolor = 0; //tinh color trung binh

for (int i=0; i<t.size(); i++)

avercolor+=t.get(i).getColor();

avercolor=avercolor/t.size();

for (int i=0; i<t.size(); i++)

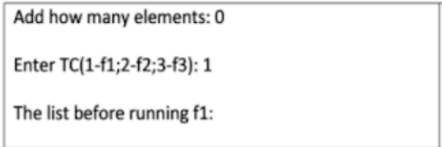
if (t.get(i).getColor()>avercolor)

count++; //neu color>trung binh thi count++

return count;

}

1. **Suppose all *owners* contain at least 2 characters. Count and return number of elements with *owner* having 2nd character (owner.charAt(1)) is a letter**





@Override

public int f1(List<Cala> t) {

int count = 0;

for (int i=0; i<t.size(); i++)

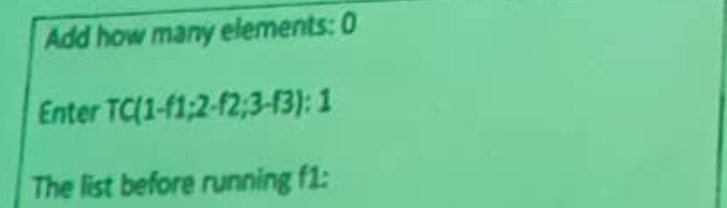
if (Character.isLetter(t.get(i).getOwner().charAt(1)))

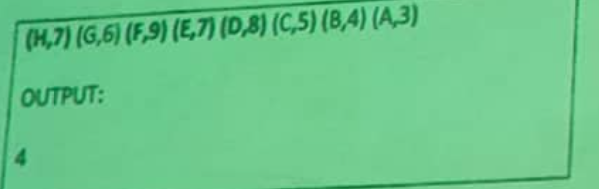
count++;

return count;

}

1. **count and return the number of *bells* with *radius* > 6**





@Override

public int f1(List<Bell> t) {

int count = 0;

for (int i=0; i<t.size(); i++)

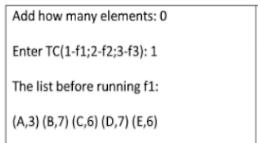
if (t.get(i).getRadius()>6)

count++;

return count;

}

1. **Return the whole part of average *rate* of all cars**



public int f1(List<Car> t) {

int sum= 0;

for (int i=0; i<t.size(); i++)

sum += t.get(i).getRate();

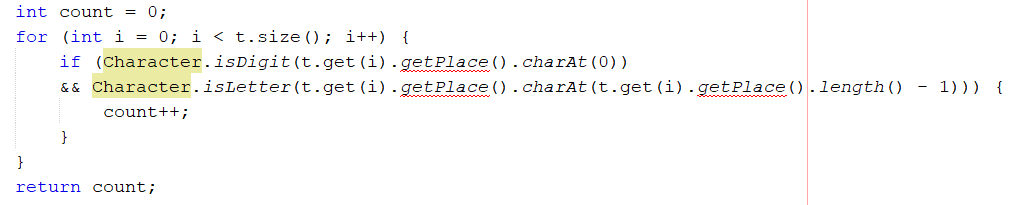
return sum/t.size();

}

//int chia int được kết quả là int => 29/5 được kết quả là int của 5.8 là 5

1. **Count and return number of elements with *place* starting with digit and ending with letter**

public int f1(List<Brick> t) {



}

## F2: Tìm kiếm rồi Remove/Swap (tìm min max rồi swap)/Move phần tử

1. Remove the first element having *color* > average color

@Override

public void f2(List<Cape> t) {

float avercolor = 0;

for (int i=0; i<t.size(); i++)

avercolor+=t.get(i).getColor();

avercolor=avercolor/t.size();

int index = -1;

for (int i = 0; i < t.size(); i++) {

if (t.get(i).getColor() > avercolor) {

index = i;

break; //neu tim dc thi dung vong lap luon

}

}

if (index != -1) { //neu chay het vong lap ma ko tim dc thi index=-1

t.remove(index);

}

}

1. remove the second element having minimum *price* sau first min(do nothing if only one minimum element in the list)

gồm 2 cái bé nhất => 3 1 4 5 6 **1** 3 5

tìm cái bé nhất min Index =>

@Override

public void f2(List<Cala> t) {

int min = t.get(0).getPrice();

int pos = 0;

for (int i = 0; i < t.size(); i++ )

{

if(t.get(i).getPrice() < min)

{

min = t.get(i).getPrice();

pos = i;

}

} //tim vi tri first min

for(int j = pos+1 ; j < t.size() ; j++) //duyet j tu sau vi tri first min

{

if(t.get(j).getPrice() == min)

{

t.remove(j);

break;

}

}

}

1. Find the first max and min *rates* in the list and swap their positions

@Override

public void f2(List<Car> t) {

Car min = Collections.min(t, new Comparator<Car>() {

@Override

public int compare(Car o1, Car o2) {

return o1.getRate()-o2.getRate();

}

});

Car max = Collections.max(t, new Comparator<Car>() {

@Override

public int compare(Car o1, Car o2) {

return o1.getRate()-o2.getRate();

}

});

int minIndex = t.indexOf(min);

int maxIndex = t.indexOf(max);

**t.set(maxIndex, min);**

**t.set(minIndex, max);**

}

1. Move the **first** *bell* with maximum *radius* to the head of the list.

//ý tưởng: tìm bell đầu tiên có radius lớn nhất, remove nó ra khỏi list rồi add lại vào đầu

//lưu ý: tìm first max thì khi so sánh luôn dùng <, không dùng <= vì last max

@Override

public void f2(List<Bell> t) {

Bell max = new Bell();

max = t.get(0);

for (int i=0; i<t.size(); i++) {

if (max.getRadius()<=t.get(i).getRadius())

max=t.get(i);

}

t.remove(max);

t.add(0, max);

}

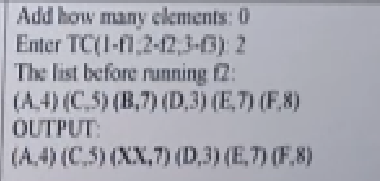
1. Find the (**first**) *brick* having maximum odd (lẻ) *price* and change its *place* to “XX”

max = get(0) = 8

list chỉ có duy nhất 1 số lẻ 5

=> max sẽ không bao giờ bằng 5

//lưu ý: phải gán max = 1 brick có price lẻ trước rồi mới tiến hành tìm max



@Override

public void f2(List<Brick> t) {

Brick max = new Brick();

max = t.get(0);

for (int i=0; i<t.size(); i++) {

if (max.getPrice()%2==1) {

max=t.get(i);

break;

}

}

for (int i=0; i<t.size(); i++) {

if (max.getPrice()<t.get(i).getPrice() && t.get(i).getPrice()%2==1) {

max = t.get(i);

}

}

max.setPlace("XX");

}

## F3: Sort

Lưu ý: //neu so sanh character, double, float, int thi dung '-'

//neu so sanh string thi dung compareTo

1. Sort the list t descendingly by the last character of *tailor*

@Override

public void f3(List<Cape> t) {

Collections.sort(t, new Comparator<Cape>() {

@Override

public int compare(Cape o1, Cape o2) {

return o2.getTailor().charAt(o2.getTailor().length()-1)

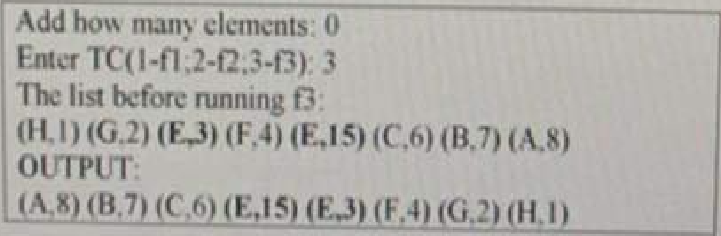
- o1.getTailor().charAt(o1.getTailor().length()-1);

}

});

}

1. Sort the list by *maker (String)* alphabetically, in case *makers* are the same, sort them descendingly by *rate (int)*

**

@Override

public void f3(List<Car> t) {

Collections.sort(t, new Comparator<Car>() {

@Override

public int compare(Car o1, Car o2) {

int a = o1.getMaker().compareTo(o2.getMaker());

if (a!=0)

return a;

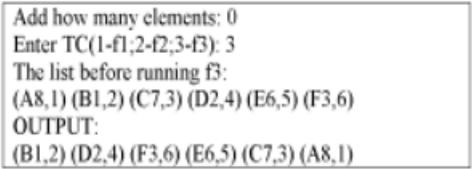
return o2.getRate()-o1.getRate();

}

});

}

1. Suppose all *owners* contains at least 2 characters. Sort the list t ascendingly by the 2nd character of *owner (String)*



@Override

public void f3(List<Cala> t) {

Collections.sort(t, new Comparator<Cala>() {

@Override

public int compare(Cala o1, Cala o2) {

return o1.getOwner().charAt(1) - o2.getOwner().charAt(1);

}

});

}

1. Suppose the list contains at least 6 elements. Sort the first 6 elements ascendingly by *radius* (other elements are unchanged), in case radii (radii is plural of radius) are the same, sort them by *maker* alphabetically

10 phần tử

6 phần tử đầu sắp xếp, 4 cái còn lại giữ nguyên

//Lưu ý: Cứ sort n phần tử liên tiếp trong 1 list, các phần tử khác giữ nguyên thì dùng sublist là nhanh nhất

@Override

public void f3(List<Bell> t) {

Collections.sort(t.subList(0, 6), new Comparator<Bell>() {

@Override

public int compare(Bell o1, Bell o2) {

int a = o1.getRadius()-o2.getRadius();

if (a!=0)

return a;

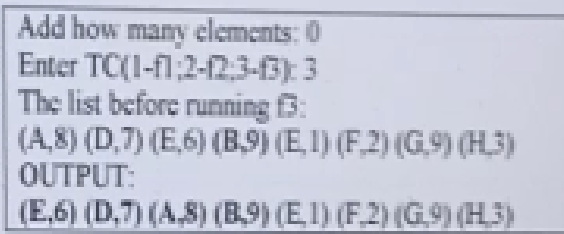
return o1.getMaker().compareTo(o2.getMaker());

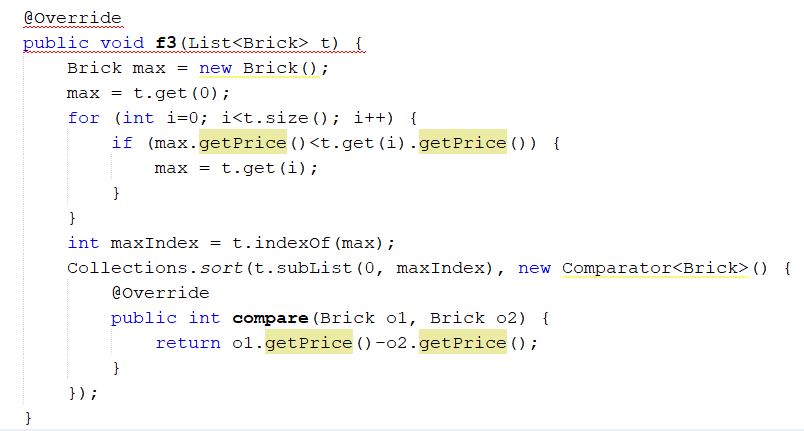
}

});

}

1. Find the (first) *brick* having maximum *price* and sort all elements before it ascendingly by *price*





@Override

public void f3(List<Brick> t) {

Brick max = new Brick();

max = t.get(0);

for (int i=0; i<t.size(); i++) {

if (max.getPrice()<t.get(i).getPrice()) {

max = t.get(i);

}

}

int maxIndex = t.indexOf(max);

Collections.sort(t.subList(0, maxIndex), new Comparator<Brick>() {

@Override

public int compare(Brick o1, Brick o2) {

return o1.getPrice()-o2.getPrice();

}

});

}

1. sort all *fans* (in the list t) ascendingly by *price*, in case their *prices* are the same, sort them ascendingly by their *code* alphabetically. The sorting must ignore case during the comparation. (không quan tâm đến viết hoa viết thường)

Abc < abc

Abc = abc

@Override

public void f3(List<Fan> t) {

Collections.sort(t, new Comparator<Fan>() {

@Override

public int compare(Fan o1, Fan o2) {

int a = o1.getPrice() - o2.getPrice();

if (a!=0)

return a;

return o1.getCode().compareToIgnoreCase(o2.getCode());

}

});

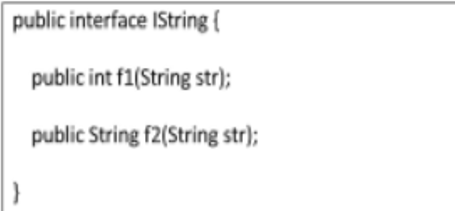
}

Q4

Đề cho sẵn interface IString, yêu cầu mình tạo class MyString implements IString.

Cả f1 và f2 đều có dạng xử lý trên String.

Lưu ý: Thỉnh thoảng cần viết thêm hàm để đỡ phức tạp



# f1

1. Return sum of odd digits in the string str

@Override

public int f1(String str) {

int sum=0;

for (int i=0; i<str.length(); i++) {

if ("13579".contains(str.charAt(i)+""))

sum+=Integer.parseInt(str.charAt(i)+"");

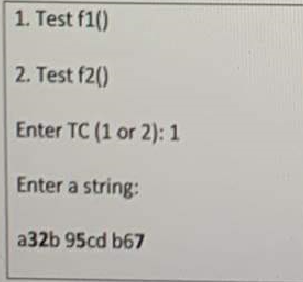
}

return sum;

}

//Lưu ý: chuyển từ String thành integer hoặc double: Integer.parseInt(str); Double.parseDouble(str) (lưu ý str là 1 string)

1. Count and return number of prime digits in str



@Override

public int f1(String str) {

int count = 0;

for (int i=0; i<str.length(); i++) {

if ("2357".contains(str.charAt(i)+""))

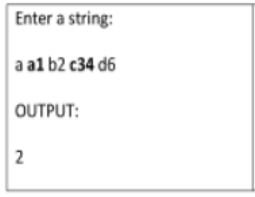
count++;

}

return count;

}

1. Count and return number of **words** containing at least 1 odd digit



@Override

public int f1(String str) {

int count = 0;

String[] list = str.split("\\s+");

for (int i=0; i<list.length; i++) {

if (list[i].contains("1") || list[i].contains("3")

||list[i].contains("5")||list[i].contains("7")

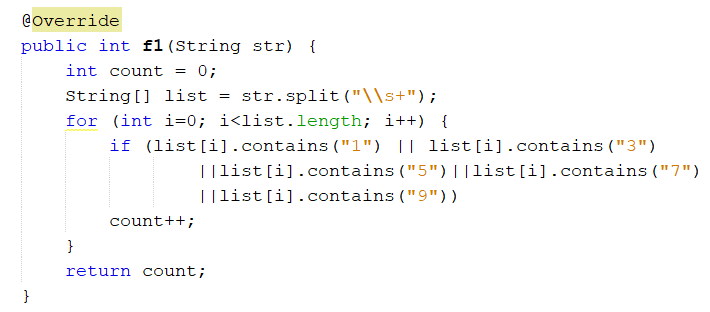
||list[i].contains("9"))

count++;

}

return count;

}



1. Count and return number of words containing at least 3 digits (word = a string without space(s))

@Override

public int f1(String str) {

int count = 0;

String[] list = str.split("\\s+");

for (int i=0; i<list.length; i++) {

if (kt(list[i]))

count++;

}

return count;

}

public static boolean kt(String s) {

int n = s.length();

int count=0;

for (int i=0; i<s.length(); i++)

if (Character.isDigit(s.charAt(i)))

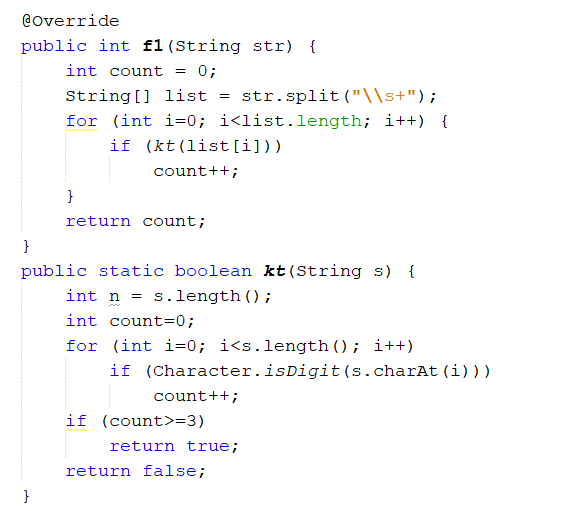
count++;

if (count>=3)

return true;

return false;

}



1. count and return number of words ending with ‘m’ or ‘n’ (not case sensitive).

Cách 1:

@Override

public int f1(String str) {

int count = 0;

String[] list = str.split("\\s+");

for (int i=0; i<list.length; i++) {

if (list[i].**endsWith**("m") || list[i].endsWith("n")

||list[i].endsWith("M") || list[i].endsWith("N"))

count++;

}

return count;

}

Cách 2:

@Override

public int f1(String str) {

int count = 0;

String[] list = str.split("\\s+");

for (int i=0; i<list.length; i++) {

if ("mnMN".contains(list[i].charAt(list[i].length()-1)+""))

count++;

}

return count;

}

1. calculate and return sum of all digits in str.

@Override

public int f1(String str) {

int sum=0;

for (int i=0; i<str.length(); i++)

if (Character.isDigit(str.charAt(i))) //

sum+= Integer.parseInt(str.charAt(i)+"");

return sum;

}

# f2

1. **return the string s, which is obtained by moving the (first) longest word in str to its end (word = a string without space(s)).**

@Override

public String f2(String str) {

String[] s = str.split("\\s+");

String max = s[0];

for (int i=0; i<s.length; i++) {

if (s[i].length()>max.length()) {

max = s[i];

}

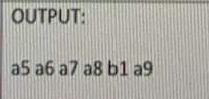
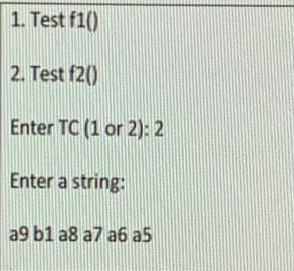
}

str = str.replaceFirst(" "+max, "");

return (str+" "+max).trim();

}

1. **Reverse order of all words in str (word = a string without space)**



@Override

public String f2(String str) {

String[] list = str.split("\\s+");

String str2 = "";

for (int i=list.length-1; i>=0; i--) {

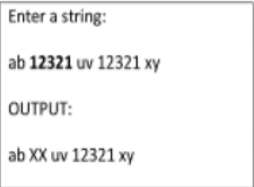
str2 += list[i] + " ";

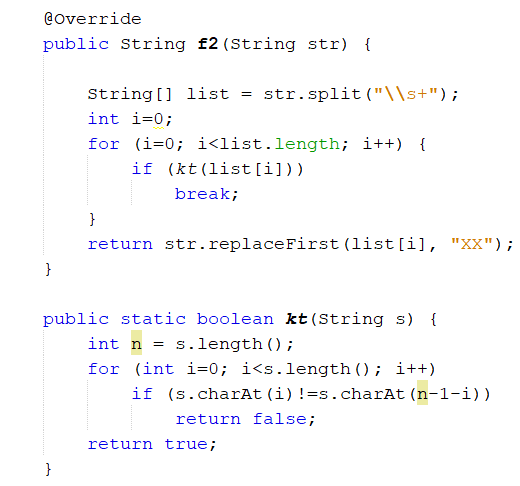
}

return str2.trim();

}

1. **Return the string s, which is obtained by replacing the (first) palindrom word in str with the string “XX” (word = a string without space(s), a word is called palindrom if it and its reverse are the same).**





@Override

public String f2(String str) {

String[] list = str.split("\\s+");

int i=0;

for (i=0; i<list.length; i++) {

if (kt(list[i]))

break;

}

return str.replaceFirst(list[i], "XX");

}

## palindrom

public static boolean kt(String s) {

int n = s.length();

for (int i=0; i<s.length(); i++)

if (s.charAt(i)!=s.charAt(n-1-i))

return false;

return true;

}

1. **remove the first alphabetically un-sorted word in the string str (e.g: if str=”ab bc de hoa ef la la” then the word hoa is that one (câu này đề chụp thiếu output nên mình đoán output là như này:**



@Override

public String f2(String str) {

String[] list = str.split("\\s+");

int i=0;

for (i=0; i<list.length; i++) {

if (!kt(list[i]))

break;

}

return str.replaceFirst(list[i]+ " ", "");

}

public static boolean kt(String s) {

int n = s.length();

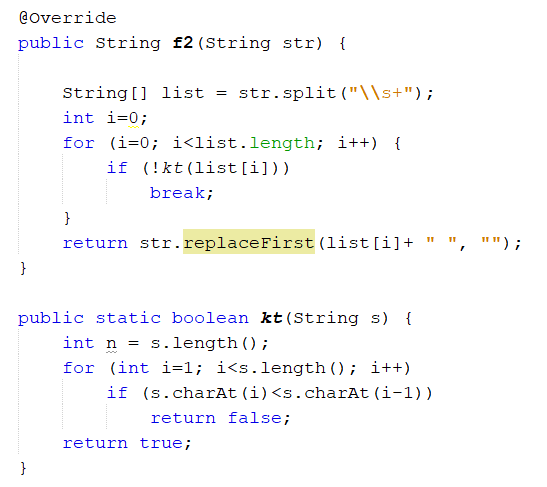
for (int i=1; i<s.length(); i++)

if (s.charAt(i)<s.charAt(i-1))

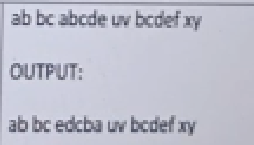
return false;

return true;

}



1. **return the string s, which is obtained by reversing the (first) longest word in the string str (word = a string without space(s)).**



@Override

public String f2(String str) {

String[] list = str.split("\\s+");

String max = list[0];

for (int i=0; i<list.length; i++) {

if (list[i].length()>max.length()) {

max = list[i];

}

}

return str.replaceFirst(max, reverse(max));

}

public static String reverse(String s) {

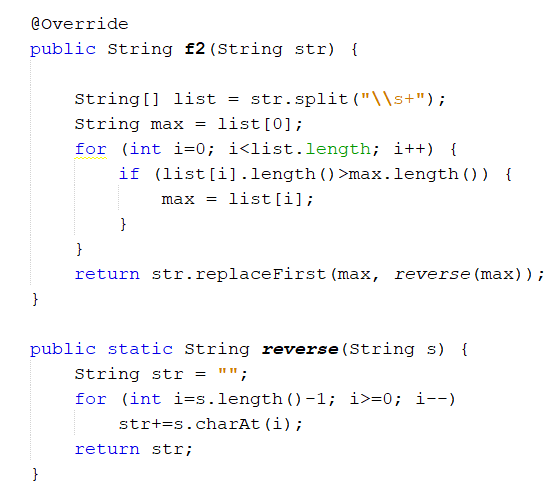
String str = "";

for (int i=s.length()-1; i>=0; i--)

str+=s.charAt(i);

return str;

}



1. **return the string s, which is obtained by reading all characters in str, if a character is a digit between 0 and 8 then increase it by 1 (others characters are unchanged). E.g., if str=”a01b2c8d9” then s = ”a12b3c9d9”**

@Override

public String f2(String str) {

String s="";

for (int i=0; i<str.length(); i++) {

if ("012345678".contains(str.charAt(i)+""))

s+=(Integer.parseInt(str.charAt(i)+"")+1);

else

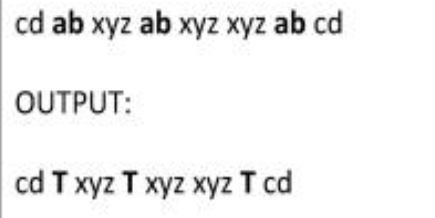
s+=str.charAt(i);

}

return s;

}

1. **suppose x is the first most frequent word in the string str. return the string s, which is obtained by replacing each word x in str with character ‘T’. (in the sample below x = “ab”)**



public String f2(String str) {

String[] list = str.split("\\s+");

String s = list[0]; //luu word co so lan xuat hien lon nhat

int max=0; //so lan xuat hien lon nhat

for (int i=0; i<list.length; i++) {

int count = 0;

for (int j=0; j<list.length; j++) {

if (list[i].equals(list[j]))

count++;

}

if (count>max) {

max = count;

s = list[i];

}

}

str = str.replace(s, "T");

return str;

}

## isPrime

public static boolean isPrime(int n){

if (n < 2) return false;

for (int i = 2; i <= Math.sqrt(n); i++) {

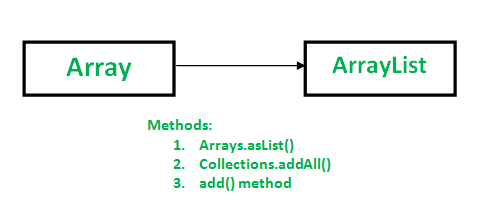
if (n%i == 0) return false;

}

return true;

}

## Array to ArrayList



## countDigit

public int countDigit(String s){

int cnt=0;

char[] arr = s.toCharArray();

for (char c : arr) {

if(Character.isDigit(c)) cnt++;

}

return cnt;

}

## find min

int min = Integer.MAX\_VALUE;

for (Train train : list) {

if (train.getLoad() < min) {

min = train.getLoad();

}

}

## decimal

Cach 1:

String.format("%2f", price)

Cach 2:

DecimalFormat df = new DecimalFormat("0.00");

df.format(price)