Klausurvorbereitungskurs

Funktionale und objektorientierte Programmierkonzepte

Svana Esche

TECHNISCHE UNIVERSITÄT DARMSTADT

Arbeitsblatt Donnerstagnachmittag
14.03.2024

Aufgabe 1: Arbeitsplan erstellen

Überlegen Sie sich, welche Themen der FOP Sie

1. bereits ausreichend beherrschen,
2. noch intensiver einüben bzw. erarbeiten müssen.

Machen Sie sich einen Plan, an welchen Tagen bis zur Klausur Sie diese Themen bearbeiten werden. Beachten Sie hierbei auch die Termine der Sprechstunden der Tutor*innen! Diese finden Sie im Übersichtsplan unter Allgemeine Informationen - Veranstalter*innen.

Auch die Themen, die Sie ausreichend beherrschen, sollten Sie bis zur Klausur noch regelmäßig wiederholen. Also müssen diese Themen ebenfalls in Ihren Arbeitsplan integriert werden.

Aufgabe 2: Arbeitsplan vervollständigen

Ergänzen Sie Ihren Arbeitsplan um *konkrete* Aufgaben. Diese können Sie aus folgenden Pool wählen:

- alle Quizze, um verschiedenste Themen aufzufrischen und wachzuhalten
- passende Aufgaben aus alten FOP-Klausuren

Aufgabe 3: Erstellen einer eigenen Exception-Klasse

Gegeben seien zwei direkt von Klasse `Exception` abgeleitete Klassen `Exc1` und `Exc2`. Dabei habe `Exc1` einem `public`-Konstruktor mit einem Parameter vom formalen Typ `String`, und `Exc2` habe einen parameterlosen `public`-Konstruktor.

> *Aufgabe:* Schreiben Sie eine `public`-Klasse `Exc3`, die direkt von `Exc1` abgeleitet ist. Klasse `Exc1` soll einen `public`-Konstruktor mit einem Parameter `a` vom formalen Typ `String` haben. Der Konstruktor von `Exc3` soll den von `Exc1` aufrufen mit dem `String` bestehend aus der Konkatenation `"Vorsicht!"` und dem Wert des aktualen Parameters `a`.

Lösungsvorschlag:

```
public class Exc3 extends Exc1 {

        public Exc3 (String a){
                super("Vorsicht!" + a);
        }

}
```

Aufgabe 4: Werfen von Exceptions

Lesen Sie die folgende Aufgabenstellung.

Aufgabenstellung

Schreiben Sie eine `public`-Klassenmethode `m` mit Rückgabetyp `String`, einem Parameter `arrStr` vom formalen Typ „Array von `String`“ und einem Parameter `n` vom formalen Typ `int`. Falls `n` keiner der Indizes von `arrStr` ist, soll `m` eine `Exc2` werfen. Ansonsten, falls der Index `n` von `b` auf kein Objekt verweist, soll `m` eine `Exc3` mit aktualem Parameter `"Kein Objekt"` werfen. Falls alle diese Fälle nicht zutreffen, wird als Ergebnis durch `m` die Referenz, das auf das Objekt an Index `n` in `arrStr` verweist, zurückgeliefert.

Die Methode `m` darf weder deklarieren, dass sie Klasse `Exc3` noch Klasse `Exception` wirft.

Erinnerung: Gegeben seien zwei direkt von Klasse `Exception` abgeleitete Klassen `Exc1` und `Exc2`. Dabei habe `Exc1` einem `public`-Konstruktor mit einem Parameter vom formalen Typ `String`, `Exc2` habe einen parameterlosen `public`-Konstruktor, und `Exc3` ist direkt von `Exc1` abgeleitet und hat einen `public`-Konstruktor mit einem Parameter vom formalen Typ `String`.

> *Aufgabe a:* Identifzieren Sie zuerst, welche Informationen für das Schreiben des Methodenkopfs und welche für den Methodenrumpf benötigt werden.

> *Aufgabe b:* Identifzieren, welche Sprachbausteine die Reihenfolge und Struktur des Methodenrumpfs bestimmen.

> *Aufgabe c:* Schreiben Sie die Methode `m` wie in der Aufgabenstellung gefordert.

Lösungsvorschlag:

Aufgabe a:

Highlighten für Methodenkopf:

Schreiben Sie eine `public`-Klassenmethode `m` mit Rückgabetyp `String`, einem Parameter `arrStr` vom formalen Typ „Array von `String`“ und einem Parameter `n` vom formalen Typ `int`. Falls `n` keiner der Indizes von `arrStr` ist, soll `m` eine `Exc2` werfen. Ansonsten, falls der Index `n` von `b` auf kein Objekt verweist, soll `m` eine `Exc3` mit aktualem Parameter `"Kein Objekt"` werfen. Falls alle diese Fälle nicht zutreffen, wird als Ergebnis durch `m` die Referenz, das auf das Objekt an Index `n` in `arrStr` verweist, zurückgeliefert.

Die Methode `m` darf weder deklarieren, dass sie Klasse `Exc3` noch Klasse `Exception` wirft.

Erinnerung: Gegeben seien zwei direkt von Klasse `Exception` abgeleitete Klassen `Exc1` und `Exc2`. Dabei habe `Exc1` einem `public`-Konstruktor mit einem Parameter vom formalen Typ `String`, `Exc2` habe einen parameterlosen `public`-Konstruktor, und `Exc3 ist direkt von Exc1 abgeleitet` und hat einen `public`-Konstruktor mit einem Parameter vom formalen Typ `String`.

Highlighten für Methodenrumpf

Schreiben Sie eine `public`-Klassenmethode `m` mit Rückgabetyp `String`, einem Parameter `arrStr` vom formalen Typ „Array von `String`“ und einem Parameter `n` vom formalen Typ `int`. Falls `n` keiner der Indizes von `arrStr` ist, soll `m` eine `Exc2` werfen. Ansonsten, falls der Index `n` von `arrStr` auf kein Objekt verweist, soll `m` eine `Exc3` mit aktualem Parameter `Kein Objekt` werfen. Falls alle diese Fälle nicht zutreffen, wird als Ergebnis durch `m` die Referenz, das auf das Objekt an Index `n` in `arrStr` verweist, zurückgeliefert.

Die Methode `m` darf weder deklarieren, dass sie Klasse `Exc3` noch Klasse `Exception` wirft.

Erinnerung: Gegeben seien zwei direkt von Klasse `Exception` abgeleitete Klassen `Exc1` und `Exc2`. Dabei habe `Exc1` einem `public`-Konstruktor mit einem Parameter vom formalen Typ `String`, `Exc2` habe einen parameterlosen `public`-Konstruktor, und `Exc3` ist direkt von `Exc1` abgeleitet und hat einen `public`-Konstruktor mit einem Parameter vom formalen Typ `String`.

Aufgabe b:

Schreiben Sie eine `public`-Klassenmethode `m` mit Rückgabetyp `String`, einem Parameter `arrStr` vom formalen Typ „Array von `String`“ und einem Parameter `n` vom formalen Typ `int`. Falls `n` keiner der Indizes von `arrStr` ist, soll `m` eine `Exc2` werfen.

⇝ if (Bedingung) Exc2 werfen

Ansonsten, falls der Index `n` von `arrStr` auf kein Objekt verweist, soll `m` eine `Exc3` mit aktualem Parameter `"Kein Objekt"` werfen.

⇝ if (Bedingung) Exc3 werfen

Falls alle diese Fälle nicht zutreffen, wird als Ergebnis durch `m` die Referenz, das auf das Objekt an Index `n` in `arrStr` verweist, zurückgeliefert.

⇝ return

Da diese Code-Zeile nur dann erreicht wird, falls die vorherige Fälle nicht zutreffen, if-else-Struktur ist aber auch möglich.

Die Methode `m` darf weder deklarieren, dass sie Klasse `Exc3` noch Klasse `Exception` wirft.

Erinnerung: Gegeben seien zwei direkt von Klasse `Exception` abgeleitete Klassen `Exc1` und `Exc2`. Dabei habe `Exc1` einem `public`-Konstruktor mit einem Parameter vom formalen Typ `String`, `Exc2` habe einen parameterlosen `public`-Konstruktor, und `Exc3` ist direkt von `Exc1` abgeleitet und hat einen `public`-Konstruktor mit einem Parameter vom formalen Typ `String`.

Aufgabe c:

```
1  public static String m (String[] arrStr, int n) throws Exc1, Exc2 {
2
3          if ( n < 0 || n >= arrStr.lenght)
4                  throw new Exc2();
5
6          if ( arrStr[n] == null)
7                  throw new Exc3 ("Kein Objekt");
8
9          return arrStr[n];
10
11 }
```

Aufgabe 5: Fangen von Exceptions

Die `public`-Objektmethode `m2` mit einem Parameter `n` vom formalen Typ `int` hat Rückgabetyp `String`. Diese Methode `m2` gehört zur selben Klasse wie Methode `m` aus Aufgabe 7 (die Klasse müssen Sie hier nicht schreiben).

In `m2` wird ein Array `c` von `String` der Länge 1 eingerichtet sowie ein neues, einzigartiges Objekt von `String` erzeugt. Ein Verweis auf dieses Objekt von `String` wird in der einzigen Komponente von `c` gespeichert.

Als nächstes wird in `m2` die Methode `m` aus Aufgabe 7 mit aktualen Parametern `c` und `n` aufgerufen. Falls dieser Aufruf eine `Exc1` wirft, soll `m2` eine `Exc2` werfen. In jedem anderen Fall soll `m2` das Ergebnis von `m` zurückliefern.

Aufgabe: Schreiben Sie die Methode `m2` wie gefordert.

Lösungsvorschlag:

```
public String m2 ( int n ) throws Exc2 {
        String[] c = new String[1];
        c[0] = new String();
        String s;

        try {
                s = m ( c, n );
        }

        catch ( Exc1 exc ) {
                throw new Exc2();
        }

        catch ( Exc2 exc ) { } // alternativ: catch ( Exc2 exc ) { return s; }
                return s;
}
```

Anmerkung `c[0] = "";` ist nicht erlaubt, da damit eventuell auf einen leeren String aus dem String Pool zurück gegriffen wird. Damit würde aber nicht wie gefordert ein neues, einzigartiges Objekt der Klasse String erzeugt.

Aufgabe 6: Drei Lösungen eines Problems - Part 1

Hier sollen Sie in den Teilaufgaben das Problem auf Listen aus der heutigen Murmelrunde auf drei verschiedene Arten analog zur Klausur lösen.

Die folgende Funktion `f` gibt das Element an Position `m` der längeren Liste zurück. Sei `m` das Minimum der Längen von `lst1` und `lst2`.

```
(define (f lst1 lst2)
        (cond
                [(empty? lst1) (first lst2)]
                [(empty? lst2) (first lst1)]
                [else (f (rest lst1) (rest lst2))]
        )
)
```

Teilaufgabe a)

Folgende Klasse kennen Sie aus der Vorlesung und den Übungen:

```
public class ListItem <T> {
        T key;
        ListItem<T> next;
}
```

> Schreiben Sie in Java die `public`-Objektmethode `f`. Die zugehörige Klasse von `f` ist generisch mit Typparameter `T` (diese Klasse brauchen Sie nicht zu schreiben). Die Listenparameter sind vom formalen Typ `ListItem<Integer>` und der Rückgabeparameter ist vom formalen Typ `Integer`.
>
> **Verbindliche Anforderung:** Die Methode `foo` ist rein rekursiv. Schleifen sind nicht erlaubt.

Teilaufgabe b)

> Schreiben Sie in Java die `public`-Objektmethode `f`. Die zugehörige Klasse von `f` ist generisch mit Typparameter `T` (diese Klasse brauchen Sie nicht zu schreiben). Die Listenparameter sind vom formalen Typ `ListItem<Integer>` und der Rückgabeparameter ist vom formalen Typ `Integer`.
>
> **Verbindliche Anforderung:** Die Methode `f` ist rein iterativ. Rekursion ist nicht erlaubt.

Teilaufgabe c)

Schreiben Sie in Java die `public`-Objektmethode `f`. Die zugehörige Klasse von `f` ist generisch mit Typparameter `T` (diese Klasse brauchen Sie nicht zu schreiben). Die Listenparameter sind vom formalen Typ `List<Integer>`. Der Rückgabeparameter ist vom formalen Typ `Integer`.

Verbindliche Anforderung: Auf die Listen `lst1` und `lst2` wird ausschließlich mit Iteratoren zugegriffen. Listen dürfen nicht in andere Datenstrukturen kopiert werden (z.B. `toArray`, `stream`).

***Erinnerung*:** Interface `List<T>` hat eine parameterlose Objektmethode `iterator` mit Rückgabetyp `Iterator<T>`. Klasse `Iterator` hat zwei parameterlose Methoden `hasNext` und `next`, wobei `hasNext` Rückgabetyp `boolean` und `next` Rückgabetyp `T` hat. Klasse `LinkedList` hat einen parameterlosen Konstruktor.

Lösungsvorschlag:

Lösung mit Rekursion:

```
public Integer f (ListItem<Integer> lst1, ListItem<Integer> lst2) {
        if ( lst1 != null && lst2 != null ) {
                return f(lst1.next , lst2.next);
        }
        if (lst1 == null)
                return lst2.key;
        return lst1.key;
}
```

Alternative Lösung mit Rekursion:

```
public Integer f (ListItem<Integer> lst1, ListItem<Integer> lst2) {
        if (lst1 == null)
                return lst2.key;
        if (lst2 == null)
                return lst1.key;
        return f(lst1.next, lst2.next);
}
```

Lösung mit Schleifen:

```
public Integer f (ListItem<Integer> lst1, ListItem<Integer> lst2) {
        Integer result;
        ListItem<Integer> p1 = lst1;
        ListItem<Integer> p2 = lst2;
        while ( p1 != null && p2 != null ) {
                p1 = p1.next;
                p2 = p2.next;
        }
        if (p1 == null)
                result = p2.key;
        else
                result = p1.key;
        return result;
}
```

Lösung mit Iteratoren:

```java
public Integer foo ( List<Integer> lst1, List<Integer> lst2 ) {
        Integer result;
        Iterator<Integer> it1 = lst1.iterator();
        Iterator<Integer> it2 = lst2.iterator();
        while ( it1.hasNext() && it2.hasNext() ) {
           it1.next();
           it2.next();
    }
    if (it1.hasNext() == false) // alternativ (!it1.hasNext())
       result = it2.next();
    else
       result = it1.next();
        return result;
}
```

Aufgabe 7: Drei Lösungen eines Problems - Part 2

In dieser Aufgabe sollen Sie ein weiteres Problem auf Listen auf drei verschiedene Arten analog zur Klausur lösen.

Gegeben sind zwei nicht-leere Listen `lst1` und `lst2` mit unterschiedlicher Länge, zurückgeliefert werden soll eine Filterung von `lst1`, und zwar alle Elemente von `lst1` die mit dem aktuell betrachteten Element von `lst2` übereinstimmen.

```
(define (foo lst1 lst2)
    (cond
        [(empty? lst1) empty]
        [(empty? lst2) empty]
        [(or (empty? (rest lst2)) (empty? (rest (rest lst2))))
            (if (equal? (first lst1) (first lst2))
                (list (first lst1))
                 empty
            )
                ]
        [(equal? (first lst1) (first lst2))
            (cons (first lst1) (foo (rest lst1) (rest (rest lst2))))]
        [else (foo (rest lst1) (rest (rest lst2)))]
        )
)
```

Teilaufgabe a)

Folgende Klasse kennen Sie aus der Vorlesung und den Übungen:

```
public class ListItem <T> {
        T key;
        ListItem<T> next;
}
```

Schreiben Sie in Java eine `public`-Objektmethode `foo`, die drei Parameter vom formalen Typ `ListItem<Integer>` hat, `lst1`, `lst2` und `result` hat.
Ihre Methode `foo` darf davon ausgehen, dass `result` auf das letzte Element einer nichtleeren Liste verweist. Die Werte, die von foo aus `lst1` gefiltert werden sollen, werden mittels `result` hinten an diese Liste angehängt und `result` zurückgegeben. Die Methode `foo` gibt nichts zurück.
Für den Test auf Gleichheit können Sie die boolesche Objektmethode `equals` von `Integer` verwenden. Sie können davon ausgehen, dass alle aktualen Parameterwerte nicht `null` sind.

Verbindliche Anforderung: Die Methode `foo` ist rein rekursiv. Schleifen sind nicht erlaubt.

Teilaufgabe b)

Schreiben Sie in Java eine `public`-Objektmethode `foo`, die drei Parameter vom formalen Typ `ListItem<Integer>` hat, `lst1`, `lst2` und `result` hat.

Ihre Methode `foo` darf davon ausgehen, dass `result` auf das letzte Element einer nichtleeren Liste verweist. Die Werte, die von foo aus `lst1` gefiltert werden sollen, werden mittels `result` hinten an diese Liste angehängt und `result` zurückgegeben. Die Methode `foo` gibt nichts zurück.
Für den Test auf Gleichheit können Sie die boolesche Objektmethode `equals` von `Integer` verwenden. Sie können davon ausgehen, dass alle aktualen Parameterwerte nicht `null` sind.

Verbindliche Anforderung: Die Methode `foo` ist rein iterativ. Rekursion ist nicht erlaubt.

Teilaufgabe c)

Schreiben Sie in Java die `public`-Objektmethode `foo`. Die zugehörige Klasse von `foo` ist generisch mit Typparameter `T` (diese Klasse brauchen Sie nicht zu schreiben). Die Listenparameter sind vom formalen Typ `List<Integer>`. Der dynamische Typ der Rückgabe soll `LinkedList<Integer>` (gemeint ist jeweils `java.util.List` bzw. `java.util.LinkedList`; Sie können voraussetzen, dass dieses Package importiert ist).

Verbindliche Anforderung: Auf die Listen `lst1` und `lst2` wird ausschließlich mit Iteratoren zugegriffen. Listen dürfen nicht in andere Datenstrukturen kopiert werden (z.B. `toArray`, `stream`).

***Erinnerung*:** Interface `List<T>` hat eine parameterlose Objektmethode `iterator` mit Rückgabetyp `Iterator<T>`. Klasse `Iterator` hat zwei parameterlose Methoden `hasNext` und `next`, wobei `hasNext` Rückgabetyp `boolean` und `next` Rückgabetyp `T` hat. Klasse `LinkedList` hat einen parameterlosen Konstruktor.

Lösungsvorschlag:

Lösung mit Rekursion:

```
public void foo ( ListItem<Integer> lst1, ListItem<Integer> lst2, ListItem<Integer>
    result ) {
        if ( lst1 == null || lst2 == null )
                return;
        if ( lst1.key.equals ( lst2.key ) ) {
                result.next = new ListItem<T>();
                result.next.key = lst1.key;
                result = result.next;
        }
        if(lst2.next == null)
                return;
        foo( lst1.next, lst2.next.next, result);
}
```

Lösung mit Schleifen:

```
public void foo ( ListItem<Integer> lst1, ListItem<Integer> lst2, ListItem<Integer>
    result) {
        ListItem<Integer> p1 = lst1;
        ListItem<Integer> p2 = lst2;
        while ( p1 != null && p2 != null ) {
                if ( p1.key.equals ( p2.key ) ) {
                        result.next = new ListItem<Integer>();
                        result = result.next;
                        result.key = p1.key;
                }
                p1 = p1.next;
                if(p2.next != null){
                        p2 = p2.next.next;
                }
                else break;
        }
}
```

Lösung mit Iteratoren:

```
public List<Integer> foo ( List<Integer> lst1, List<Integer> lst2 ) {
        Iterator<Integer> it1 = lst1.iterator();
        Iterator<Integer> it2 = lst2.iterator();
        List<Integer> result = new LinkedList<Integer>();

        while ( it1.hasNext() && it2.hasNext()) {
                Integer i = it1.next();
                if ( i.equals ( it2.next()  )) {
                        result.add ( i );
                }
                if(it2.hasNext())
                        it2.next();
                else {
                        break;
                }
        }
        return result;
}
```

Aufgabe 8: Array mit Zahlen

Schreiben Sie eine `public`-Klassenmethode `foobar`, die einen Parameter `a` vom Typ „Array von `double`“ und Rückgabetyp „Array von `int`“ hat (die Klasse, zu der `foobar` gehört, müssen Sie nicht schreiben). Ihre Methode `foobar` kann ohne Prüfung davon ausgehen, dass `a` tatsächlich auf ein Arrayobjekt verweist.

Die Länge des zurückgelieferten Arrays soll gleich der Anzahl derjenigen Indizes `i>0` im Indexbereich von `a` sein, bei denen der Wert an `i` einer ganzen Zahl entsprechen würde. Für jedes solche `i` soll die letzte Ziffer der entsprechenden Ganzzahl im zurückgelieferten Array sein, und zwar in aufsteigender Reihenfolge der `i`.

***Beispiel 1*:** $[\underline{1}.0,\ 3.3,\ 22.5,\ 12\underline{9}.0,\ \underline{6}.0] \longrightarrow [1,\ 9,\ 6]$

***Beispiel 2*:** $[6\underline{7}.0,\ 3.3,\ 22.5,\ 19.1,\ 6.9] \longrightarrow [7]$

Verbindliche Anforderung: Es darf keine Funktionalität aus der Java-Standardbibliothek verwendet werden (das betrifft insbesondere, aber nicht nur Streams sowie Klassen, die Interface `List` implementieren).

Lösungsvorschlag:

```java
public static int[] foobar (double[] a){
        int laenge = 0;

        for(int i = 0; i < a.length; i++){
                if ((int)a[i] == a[i]){
                        laenge++;
                }
        }

        int[] b = new int [laenge];
        int indexB = 0;

        for(int i = 0; i < a.length; i++){
                if ((int)a[i] == a[i]){
                        b[indexB] = (int) a[i] % 10;
                        indexB++;
                }
        }
        return b;
}
```

Alternative Lösung:

```java
public static int[] foobar (double[] a){
        int laenge = 0;

        for(int i = 0; i < a.length; i++){
                int intOfDouble = (int) a[i];
                if (a[i] - (double) intOfDouble == 0){
                        laenge++;
                }
        }

        int[] b = new int [laenge];
        int indexB = 0;

        for(int i = 0; i < a.length; i++){
                int intOfDouble = (int) a[i];
                if (a[i] - (double) intOfDouble == 0){
                        b[indexB] = intOfDouble % 10;
                        indexB++;
                }
        }

        return b;
}
```

Alternative Lösung mit Rekursion:

```java
public static int[] foobar(double[] a){
        int numberOfIndizes = getNumberOfIndizes(a, 0);
        int[] result = new int [numberOfIndizes];
        fillArray(a, result, 0, 0);
        return result;
}

public static int getNumberOfIndizes (double[] a, int index){
        if (a[index] == null)
                return 0;
        int found = 0;
        if ((int) a[index] = a[index])
                found = 1;
        else
                found = 0;
        return found + getNumberOfIndizes(a, index + 1);
}

public static void fillArray (double[] a, int[] result, int indexA, int indexResult){
        if ((int) a[indexA] = a[indexA]){
                result[indexResult] = (int) a[indexA] % 10;
                indexResult++;
        }
        fillArray(a, result, indexA + 1; indexResult);
}
```

Aufgabe 9: Array mit Buchstaben

Schreiben Sie eine `public`-Klassenmethode `foobar`, die einen Parameter `a` vom Typ „Array von `char`“ und Rückgabewert `m` vom Typ `HashMap` hat (die Klasse, zu der `foobar` gehört, müssen Sie nicht schreiben). Hierbei sind die Schlüsselwerte vom formalen Typ `char` und die zugehörige Werte vom formalen Typ `int`. Ihre Methode `foobar` kann ohne Prüfung davon ausgehen, dass `a` tatsächlich auf ein Arrayobjekt verweist.

In `m` soll für jeden Wert von `char` als *Key*, der mindestens ein Vorkommen in `a` hat, dessen absolute Häufigkeit, also dessen Anzahl als *Value* gespeichert werden.

***Beispiel*:**

$\{a, T, b, a, b, T, a, a\} \rightarrow$

Key	Value
a	4
T	2
b	2

Verbindliche Anforderung: Es darf keine Funktionalität aus der Java-Standardbibliothek verwendet werden (das betrifft insbesondere, aber nicht nur Streams sowie Klassen, die Interface `List` implementieren).

Lösungsvorschlag:

```
public static HashMap<Character, Integer> foobar (char[] a){
        HashMap<Character, Integer> m = new HashMap<Character, Integer>();

        for(char c : a){
                int countC = 1;
                if (m.containsKey((Character) c)){
                        countC = m.get((Character) c) + 1;
                }
                m.put((Character) c, countC );
        }
        return m;
}
```

Alternative Lösung:

```
public static HashMap<Character, Integer> foobar (char[] a){
        HashMap<Character, Integer> m = new HashMap<Character, Integer>();

        for(char c : a){
                m.put(cChar, m.getOrDefault(cChar, 0) + 1);
        }
        return m;
}
```