

Hinweise zur Probeklausur

- Wir empfehlen, die Probeklausur unter so realistischen Bedingungen wie möglich zu schreiben. Das heißt, Sie sollten sich 120 Minuten möglichst **ungestört** mit der **ausgedruckten** Probeklausur beschäftigen und nur die für die echte Klausur erlaubten Hilfsmittel verwenden.
- Die Punkte pro Aufgabe entsprechen etwa dem Zeitaufwand in Minuten. Falls Sie bei einer Teilaufgabe nicht weiterkommen, überspringen Sie erstmal diese Teilaufgabe; ansonsten verlieren Sie in der echten Klausur unnötig Zeit.
- Sie dürfen bei jeder Aufgabe auf andere Methoden in der Aufgabe zurückgreifen, die sie selbst in der Aufgabe implementieren müssen oder vorgegeben sind. Ersteres ist auch dann erlaubt, falls sie diese Teilaufgabe nicht bearbeitet haben.
- Deckblatt, Hinweise usw. entsprechen voraussichtlich schon denen der echten Klausur. Sie können sich also bereits jetzt damit beschäftigen und alle Fragen dazu klären, dann müssen Sie das Deckblatt in der tatsächlichen Klausur nur noch überfliegen.
- Die Klausur wird doppelseitig gedruckt und oben links getackert. Zusammenhängende Aufgaben werden dabei nach Möglichkeit (wie auch in der Probeklausur) beim Blättern nebeneinander zu liegen kommen.
- Die Probeklausur wird in der letzten Woche in den Übungen besprochen. Außerdem gibt es voraussichtlich ein Lösungs-Video.
- In der Probeklausur können teilweise Abwandlungen alter Klausur- und Übungsaufgaben vorkommen. Dies kann in einer echten Klausur ebenfalls der Fall sein, muss es aber nicht.
- Bei jeder Klausur muss eine Auswahl aus den bearbeiteten Themen getroffen werden. Die Auswahl der Themen und die Verteilung der Punkte auf die einzelnen Themen kann in der echten Klausur anders aussehen. Insbesondere werden die echten Klausuren auch in kleinen Aufgaben die Themen der letzten beiden Vorlesungswochen aufgreifen (siehe Selbsttests zu den Wochen 14 und 15 im Ilias).
 - Folgende Themen kommen beispielsweise in dieser Probeklausur nicht vor, sind aber klausurrelevant (unvollständige Aufzählung): Überladen, Suche, Klassenvariablen, Generics, abstrakte Klassen, super
- Auch in der echten Klausur wird die letzte Aufgabe aus dem Themenbereich der objekt-orientierten Programmierung am meisten Punkte geben.
- Sie können die Probeklausur nutzen, um herauszufinden, mit welchen Aufgabentypen Sie besonders gut zurechtkommen; diese Aufgaben wären ein guter Startpunkt in der echten Klausur. Wenn Sie mit bestimmten Aufgabentypen Schwierigkeiten zu haben, versuchen Sie diese zu identifizieren und Unklarheiten zu beseitigen.
- Wir bemühen uns, dass die Probeklausur eher schwieriger und zeitlich knapper kalkuliert ist als Haupt- und Nachklausur, und dass Haupt- und Nachklausur gleich schwierig sind. Da der Schwierigkeitsgrad aber keine objektive Einschätzung ist, sondern von den eigenen Stärken und Schwächen abhängt, können wir das nicht garantieren. Die Klausuren orientieren sich an den im Semester bearbeiteten Materialien und den Lernzielen der Vorlesung.
- Zum weiteren Üben eignen sich insbesondere die jüngeren Altklausuren, die Sie im Klausurarchiv der Fachschaft Informatik¹ finden.

¹<https://fscs.hhu.de/klausur-archiv/>

Januar 2023

Probeklausur
Programmierung
WS 22/23

Nachname: _____ Vorname: _____

Matrikelnummer: _____ Sitzplatznummer: _____

Zusätzliche Blätter: _____ Unterschrift: _____

Hinweise:

- Diese Klausur enthält 22 nummerierte Klausurseiten. Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält. Sie dürfen die Heftung der Klausur nicht auftrennen.
- Sie erhalten außerdem von uns leere Blätter. Sollten Sie auf diesen Blättern für die Korrektur relevante Teile Ihrer Lösung notieren, markieren Sie dies deutlich an der entsprechenden Aufgabe und auf dem zusätzlichen Blatt. Schreiben Sie auf alle zusätzlichen Blätter Ihren Namen. Geben Sie außerdem oben auf dem Deckblatt an, wie viele zusätzliche Blätter Sie zur Korrektur abgeben. Wenn Sie weiteres Papier benötigen, melden Sie sich bitte.
- Alle Fachbegriffe in dieser Klausur werden wie in der Vorlesung definiert verwendet. Alle Fragen beziehen sich auf die in der Vorlesung vorgestellte Java-Version 17. Programmcode muss in Java geschrieben werden.
- Antworten dürfen auf Deutsch oder Englisch gegeben werden. Sofern keine ausformulierten Sätze verlangt sind, reichen nachvollziehbare Stichworte als Antwort.
- Zugelassene Hilfsmittel: eine beidseitig beschriebene oder bedruckte DIN-A4-Seite, Wörterbuch (Wörterbücher müssen vor Beginn der Klausur den Aufsichtspersonen zur Kontrolle vorgelegt werden.)
- Schalten Sie Ihr Mobiltelefon aus. Täuschungsversuche führen zum sofortigen Ausschluss von der Klausur. Die Klausur wird dann als nicht bestanden gewertet.
- Schreiben Sie **nicht** mit radierbaren Stiften und auch **nicht** mit rot!

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	8	Σ
Punktzahl	10	6	2	9	22	6	6	29	90
Erreicht									

Aufgabe 1

 / 10 Punkte

- (a) [6 Punkte] Gegeben seien die folgenden Klassen:

```
Person.java Java
1 public class Person {
2     private String name;
3
4     public Person(String name) {
5         this.name = name;
6     }
7
8     @Override
9     public String toString()
10    return name;
11 }
12 }
```



```
Studi.java Java
1 public class Studi extends Person {
2     private int nummer;
3
4     public Studi(String name, int nummer) {
5         super();
6         this.nummer = nummer;
7     }
8
9     @Override
10    public String toStirng() {
11        return super.toString() + ", " + nummer;
12    }
13 }
```

Beim Compilieren der Klassen gibt es folgende Fehlermeldungen:

```
• • •

./Person.java:9: error: ';' expected
    public String toString()
               ^
./Person.java:12: error: class, interface, or enum expected
}
^
./Person.java:9: error: missing method body, or declare abstract
    public String toString()
               ^
Studi.java:5: error: constructor Person in class Person cannot be applied to given
    types;
        super();
               ^
required: String
found: no arguments
reason: actual and formal argument lists differ in length
Studi.java:9: error: method does not override or implement a method from a
    supertype
    @Override
               ^
5 errors
```

Geben Sie an, in welchen Klassen und Zeilen die **Ursachen** für die Fehler sind, beschreiben Sie die Fehler jeweils kurz (max. 1 Satz) und geben Sie die korrigierte Codezeile **vollständig** an, sodass der Code das tut, was bei der Programmierung wahrscheinlich vorgesehen war. Geben Sie keine Folgefehler an, die durch Korrektur eines vorherigen Fehlers behoben werden.

Klasse, Zeilennummer: _____

Fehlerbeschreibung: _____

Korrektur: _____

- (b) [2 Punkte] Ihr Terminal sieht gerade wie folgt aus:

```
● ● ●
/home/jan/projects %
```

Sie wollen die Klasse `Hello` im Ordner `/home/jan/projects/blatt01` kompilieren und die enthaltene `main`-Methode ausführen. Welche Befehle müssen Sie dazu eingeben? *Geben Sie nur die für das Kompilieren und Ausführen notwendigen Befehle an.*

- (c) [2 Punkte] Wir haben die Noten einer Klausur in der Textdatei `noten` gespeichert. Außerdem haben wir ein bereits kompiliertes Java-Programm `Schnitt`, das den Durchschnitt von Zahlen berechnen und ausgeben kann.

```
● ● ●
% ls
noten Schnitt.class Schnitt.java
% cat noten
1,7 1,3 1,7 2,3 2,3 1 4 3 5 3,7

Schnitt.java
```

Java

```
import java.util.Scanner;
import java.util.LinkedList;
import java.util.List;

public class Schnitt {
    public static void main(String[] args) {
        Scanner eingabe = new Scanner(System.in);
        List<Double> zahlen = new LinkedList<>();

        // liest Zahlen ein
        while(eingabe.hasNext()) {
            double zahl = eingabe.nextDouble();
            zahlen.add(zahl);
        }

        // berechnet den Durchschnitt (den Code müssen Sie nochvollziehen)
        double schnitt = zahlen.stream()
            .mapToDouble(Double::doubleValue)
            .average()
            .orElse(Double.NaN);

        // gibt Durchschnitt aus
        System.out.println(schnitt);
    }
}
```

Geben Sie einen Aufruf des Programms `Schnitt` an, sodass der Durchschnitt der Zahlen in der Datei `noten` berechnet und das Ergebnis in die Datei `schnitt` geschrieben wird.

Aufgabe 2

 / 6 Punkte

Formen Sie die Kontrollstrukturen in den folgenden Codeausschnitten um, sodass sich die Semantik des Codes nicht ändert. Benutzen Sie in Ihrem Code jeweils nur die von der Aufgabe vorgegebene Art von Kontrollstruktur.

- (a) [2 Punkte] Formen Sie die folgende while-Schleife in eine do-while-Schleife um. Halten Sie die Anzahl der Stellen, an denen `Math.random()` aufgerufen wird, minimal.

Vorgabe**Java**

```
double z = Math.random();
System.out.print(z);
while(z <= 0.5) {
    z = Math.random();
    System.out.print(z);
}
System.out.print("Erfolg");
```

Ihre Lösung**Java**

```
System.out.print("Erfolg");
```

- (b) [2 Punkte] Formen Sie die folgende for-each-Schleife in eine for-Schleife um.

Vorgabe**Java**

```
String[] names = {"Alice", "Bob", "Eve"};
int position = 1;
for(String name: names) {
    System.out.print(position + ": " + name);
    position++;
}
```

Ihre Lösung**Java**

```
String[] names = {"Alice", "Bob", "Eve"};
```

(c) [2 Punkte] Formen Sie die folgende if-Verzweigung in eine switch-Verzweigung um:

Vorgabe

Java

```
int taste = 2;
String preis;
if(taste == 1 || taste == 3) {
    preis = "1 €";
} else if(taste == 2 || taste == 4) {
    preis = "2 €";
} else {
    preis = "err";
}
System.out.print(preis + ", bitte");
```

Ihre Lösung

Java

```
int taste = 2;
String preis;
```

```
System.out.print(preis + ", bitte");
```

Aufgabe 3

 / 2 Punkte

Angenommen, Sie müssten öfter Arrays mit 1 Millionen Matrikelnummern sortieren. Würden Sie dafür eine Sortiermethode benutzen, die Insertion Sort verwendet? Begründen Sie Ihre Antwort in 1–2 ausformulierten Sätzen und schlagen Sie ggf. eine Alternative vor.

Aufgabe 4

 / 9 Punkte

Die n -te Lucas-Zahl $L(n)$ ist wie folgt rekursiv definiert:

$$L(n) = \begin{cases} 2 & \text{wenn } n = 0 \\ 1 & \text{wenn } n = 1 \\ L(n - 1) + L(n - 2) & \text{wenn } n > 1 \end{cases}$$

Die ersten Lucas-Zahlen sind $2, 1, 3, 4, 7, 11, \dots$

Schreiben Sie ein Java-Programm **[Lucas]**, das eine ganze Zahl x als Konsolenargument entgegennimmt und die Lucaszahlen von $L(0)$ bis $L(x)$ auf der Standardausgabe ausgibt. Falls **kein Argument** oder **keine ganze Zahl** übergeben wird, soll sich das Programm mit der Ausgabe **[err]** beenden. Falls Sie Exceptions abfangen, fangen Sie diese so genau wie möglich ab (fangen Sie also nicht alle möglichen Exceptions).

*Zur Erinnerung: Die Parse-Methoden werfen im Fehlerfall eine **[NumberFormatException]**.*

Beispiel-Aufrufe:

```
% java Lucas 4
2
1
3
4
7
% java Lucas -3
% java Lucas
err
% java Lucas vier
err
```

Lucas.java

Java

Lucas.java (Fortsetzung)

Java

Aufgabe 5

 / 22 Punkte

Aus der Vorlesung kennen Sie folgende Implementierung von Insertion Sort, die ein Array von Integern aufsteigend sortiert:

```
Java
public static void sort(int[] numbers) {
    for(int currentIndex = 1; currentIndex < numbers.length; currentIndex++) {
        int currentNumber = numbers[currentIndex];
        int insertionPosition = currentIndex;
        while(insertionPosition > 0 && numbers[insertionPosition - 1] > currentNumber) {
            numbers[insertionPosition] = numbers[insertionPosition - 1];
            insertionPosition--;
        }
        numbers[insertionPosition] = currentNumber;
    }
}
```

Gegeben sei die folgende Klasse `Produkt`, die eine Produkt mit Namen, Preis und Verfügbarkeit repräsentiert.

```
Java
Produkt.java
public class Produkt {
    private final int preis; // in Cent
    private boolean verfuegbar;
    private final String name;

    public Produkt(int preis, String name) {
        this.preis = preis;
        this.name = name;
        this.verfuegbar = true;
    }

    public void nichtMehrVerfuegbar() {
        this.verfuegbar = false;
    }

    public boolean istVerfuegbar() {
        return verfuegbar;
    }

    public int getPreis() {
        return preis;
    }

    public String toString() {
        return name + ": " + preis + " Cent";
    }
}
```

- (a) [6 Punkte] Vervollständigen Sie die Klassenmethode `nachPreis`, die ein Array von `Produkt`-Objekten übergeben bekommt und ein neues Array zurückgeben soll, in dem dieselben Objekte **aufsteigend** nach ihrem Preis sortiert sind; die Reihenfolge der Objekte im übergebenen Array soll dabei von der Methode **nicht** verändert werden; das Original-Array und das sortierte Array sollen dieselben Objekte im Heap referenzieren.

Falls der Methode `null` übergeben wird oder ein Element im Array `null` ist, soll eine `IllegalArgumentException` geworfen werden.

```
Produkt.java (Fortsetzung) Java

public _____ nachPreis(Produkt[] produkteOrig) {
    if(produkteOrig == _____) {
        _____;
    }
    Produkt[] produkte = new Produkt[_____];
    for(int i = 0; i < produkte.length; i++) {
        if(_____) {
            // Exception werfen
            _____;
        }
        _____;
    }
    for(int i = 1; i < _____; i++) {
        _____;
        int einfPos = i;
        while(einfPos > 0
              && _____) {
            _____;
            einfPos--;
        }
        _____;
    }
    _____;
}
```

Es soll nun eine nach Preis aufsteigend sortierte Auflistung von günstigen Produkten erstellt werden, wobei nur verfügbare Produkte ausgegeben werden.

Ergänzen Sie die Klasse **Auswertung** um folgende **private, statische** Methoden. Sie können immer davon ausgehen, dass kein Array und kein Array-Wert **null** ist.

- (b) [5 Punkte] **Produkt [] verfuegbare(Produkt [])**: Gibt ein neues Produkt-Array zurück, das nur genau die Produkte aus dem übergebenen Array enthält, die verfügbar sind.
- (c) [5 Punkte] **Produkt [] guenstige(Produkt [])**: Gibt ein neues Produkt-Array zurück, das nur die Produkte aus dem übergebenen Array enthält, die **weniger** als 80 Cent kosten.
- (d) [2 Punkte] **void ausgeben(Produkt [])**: Gibt die String-Repräsentation der übergebenen Produkte auf der Standardausgabe aus (lässt Reihenfolge unverändert)
- (e) [4 Punkte] Vervollständigen Sie die **main**-Methode, sodass die String-Repräsentationen aller bereits angelegten Produkte, die verfügbar sind und weniger als 80 Cent kosten, nach Preis aufsteigend sortiert ausgegeben werden. Sie müssen dabei alle in dieser Aufgabe geschriebenen Methoden verwenden; vergessen Sie nicht, dass **nachPreis** in einer anderen Klasse steht. Der Code muss auch dann korrekt funktionieren, wenn die Eigenschaften der drei vorgegebenen Produkt-Objekte anders wären.

Auswertung.java

```
public class Auswertung {
```

Java

Auswertung.java (Fortsetzung)

Java

```
public static void main(String[] args) {  
    Produkt apfel = new Produkt(70, "Apfel");  
    Produkt birne = new Produkt(80, "Birne");  
    Produkt lauch = new Produkt(60, "Lauch");  
  
    birne.nichtMehrVerfuegbar();  
  
}  
}
```

Aufgabe 6

_____ / 6 Punkte

Gegeben sei die folgende Klasse **[StringList]**, die eine einfach verkettete Liste implementiert, in der Strings gespeichert werden können:

```
StringList.java
Java
public class StringList {
    private class Node {
        private String data;
        private Node next;

        private Node(String data, Node next) {
            this.data = data;
            this.next = next;
        }
    }

    private Node head;

    public StringList(String[] initialValues) {
        for(int i = initialValues.length - 1; i >= 0; i--) {
            head = new Node(initialValues[i], head);
        }
    }

    public String toString() {
        // ... (nicht abgedruckt)
    }

    public void removeAll(String needle) {
        if(head != null && head.data.equals(needle)) {
            head = head.next;
        }
        Node current = head;
        while(current != null) {
            if(current.next != null && current.next.data.equals(needle)) {
                current.next = current.next.next;
            }
            current = current.next;
        }
    }
}
```

Die Methode **void removeAll(String)** soll alle Strings aus der Liste entfernen, die inhaltlich gleich dem übergebenem String sind. Die Methode funktioniert aber nicht richtig; bei den folgenden Testaufrufen gibt es jeweils nach den letzten drei Aufrufen nicht die erwartete Ausgabe:

```
Test.java
Java
public class Test {
    public static void main(String[] args) {
        String[] listElements = {"a", "a", "b", "c", "c", "d", "c", "e", "e", "e"};
        StringList list = new StringList(listElements);
        System.out.println(list); // erwartet: a, a, b, c, c, d, c, e, e, e,
        list.removeAll("b");
        System.out.println(list); // erwartet: a, a, c, c, d, c, e, e, e,
        list.removeAll("c");
        System.out.println(list); // erwartet: a, a, d, e, e, e,
        list.removeAll("a");
        System.out.println(list); // erwartet: d, e, e, e,
        list.removeAll("e");
        System.out.println(list); // erwartet: d,
```

```
% java Test  
a, a, b, c, c, d, c, e, e, e,  
a, a, c, c, d, c, e, e, e,  
a, a, c, d, e, e, e,  
a, c, d, e, e, e,  
a, c, d, e,
```

Geben Sie eine korrigierte Implementierung der Methode an; alternativ können Sie auch **eindeutig** beschreiben, wie die fehlerhafte Implementierung korrigiert werden kann:

Ihre Lösung

Java

```
public void removeAll(String needle) {
```

```
}
```

Aufgabe 7

 / 6 Punkte

Gegeben sei die Klasse `Tree` für einen binären Suchbaum, in dem Doubles gespeichert werden können:

```
Tree.java  
Java  
1 public class Tree {  
2     private class BinaryNode {  
3         private double element;  
4         private BinaryNode left, right;  
5         private BinaryNode(double element) {  
6             this.element = element;  
7         }  
8     }  
9     private BinaryNode root;  
10    public void insert(double newNumber) {  
11        // ... (Implementierung nicht abgedruckt)  
12    }  
13}
```

Vervollständigen Sie die Methode `double sumSmall()`, die die **Summe** aller Einträge im Baum zurückgibt, die **größer oder gleich 0 und kleiner** als 100 sind; bei einem leeren Suchbaum ist die Summe gleich 0. Nutzen Sie in Ihrem Code die Eigenschaften eines binären Suchbaumes aus, um die Anzahl der betrachteten Knoten minimal zu halten. Sie dürfen zusätzliche Hilfsmethoden mit minimaler Sichtbarkeit schreiben.

```
Tree.java (Fortsetzung)  
Java  
public double sumSmall() {
```

Tree.java (Fortsetzung)

Java

```
}
```

Aufgabe 8

 / 29 Punkte

In dieser Aufgabe sollen Sie Interfaces, Klassen und Methoden für einen Kassenautomaten eines Schwimmbads programmieren.

Hinweise:

- Sie dürfen alle Variablennamen frei wählen.
- Wählen Sie sinnvolle Datentypen für Ihre Variablen.
- Alle Instanzvariablen müssen **privat** sein.
- Wenn kein Konstruktorverhalten vorgeschrieben ist, reicht der Default-Konstruktor.
- Das genaue Format von Textausgaben ist Ihnen überlassen.
- Innerhalb dieser Aufgabe müssen Sie keine Exceptions abfangen. Sie müssen keine Parameter validieren.
- Lesen Sie sich die Aufgabenstellung vor Beginn der Implementierung **vollständig** durch, um einen besseren Überblick über das Gesamtbild zu erhalten.
- Gehen Sie davon aus, dass alle in dieser Aufgabe genannten Klassen und Interfaces im selben Package liegen.

(a) [1½ Punkte]

Schreiben Sie ein **öffentliches** Interface **Wetter**, das eine Methode **temperatur()** ohne Parameter vorschreibt; diese Methode soll später die aktuelle Temperatur zurückgeben.

Wetter.java

Java

(b) [1½ Punkte]

Schreiben Sie ein **öffentliches** Interface `Ticket`, das eine Methode `int preis()` ohne Parameter vorschreibt, die später den Ticketpreis in Cent zurückgibt.

`Ticket.java`

Java

(c) [3 Punkte]

Schreiben Sie eine nicht abstrakte, **öffentliche** Klasse `FakeWetter`, die das `Wetter`-Interface sinnvoll implementiert. Der Konstruktor bekommt eine Temperatur übergeben. Die `temperatur`-Methode gibt immer diese Temperatur zurück.

`FakeWetter.java`

Java

(d) [7 Punkte]

Erstellen Sie nicht abstrakte, **öffentliche** Klassen `ErwachsenenTicket` und `KinderTicket`, die das `Ticket`-Interface sinnvoll implementieren. Jedes Erwachsenen-Ticket kostet 300 Cent, jedes Kinder-Ticket 150 Cent. Überschreiben Sie außerdem die `toString`-Methode, sodass jeweils die Art des Tickets (Erwachsene/Kinder) und der Preis zurückgegeben werden.

`ErwachsenenTicket.java`

Java

`KinderTicket.java`

Java

(e) [12 Punkte]

Vervollständigen Sie die Klasse `Quittung`. Eine Quittung speichert eine `Wetter`-Instanz und ein Array von Tickets. Der **öffentliche Konstruktor**

`Quittung(int, int, Wetter)` bekommt eine Anzahl von Erwachsenen-Tickets, eine Anzahl von Kinder-Tickets und eine `Wetter`-Instanz übergeben; er legt entsprechend viele Tickets im `Ticket`-Array ab.

Schreiben Sie eine **öffentliche** Methode `int gesamtpreis()`, die den Gesamtpreis aller gespeicherten Tickets zurückgibt. Es gibt einen Rabatt von 200 Cent, wenn der Gesamtpreis (ohne Rabatt) größer als 1000 Cent und die Temperatur (wie vom `Wetter`-Objekt angegeben) größer als 30 ist.

Überschreiben Sie die `toString`-Methode, sodass die String-Repräsentationen aller Tickets und der Gesamtpreis zurückgegeben werden. (Rückseite beachten)

Quittung.java

Java

```
public class Quittung {
```

Quittung.java (Fortsetzung)

Java

}

(f) [4 Punkte]

Ergänzen Sie die `main`-Methode der Klasse `Kassenautomat`. Die Methode nimmt das eingeworfene Geld (in Cent), die Anzahl der Erwachsenen- und Kinder-Tickets entgegen und soll dann folgendes tun:

1. Eine Instanz von `FakeWeather` mit Temperatur 30 wird erstellt.
2. Eine Quittung wird erstellt.
3. Wenn genug Geld eingeworfen wurde, um die Quittung zu bezahlen, wird die String-Präsentation der Quittung ausgegeben.
4. Andernfalls wird `zu wenig Geld` ausgegeben

Kassenautomat.java

Java

```
public class Kassenautomat {
    public static void main(String[] args) {
        int geldGegeben = Integer.parseInt(args[0]);
        int anzahlErwachsene = Integer.parseInt(args[1]);
        int anzahlKinder = Integer.parseInt(args[2]);

        Wetter w = _____;
        Quittung q = _____;

        if(_____) {
            _____;
        } else {
            _____;
        }
    }
}
```