PROGRAMMIERUNG 2 ALTKLAUSUR

AUFGABE 1

A) Nennen sie 2 Vorteile und 2 Nachteile einer rekursiven Implementierung gegenüber einer Iterativen Implementierung.

Vorteil	Nachteil
Kürzere Formulierung	Schlechtere Laufzeit
Leicht verständliche Lösung	Höhere Speicherkomplexität

B) Wo Werden bei rekursiven Implementierungen die Zwischenergebnisse gespeichert.
Auf dem Stack.

C) Eine Rekursion kann immer auch Iterativ berechnet werden.
Ja.

D) Erläutern sie warum eine Rekursive Implementierung der Fibonacci Folge sehr ineffizient ist. Wie kann die Berechnung effizienter gestaltet werden.

- Redundante Berechnung.
- Verbraucht eine Menge an Speicher.
- Sie kann durch dynamische Programmierung effizienter gestaltet werden.

E) Welche 4 Typen von inneren Klassen gibt es.

- Static Nested
- Inner Class
- Local Class
- Anonymous Class

F) Was ist eine Meta-Annotation?
Eine Annotation die auf eine andere Annotation angewandt wird.

G) Erläutern sie was Generische Programmierung ist und welche Vorteile diese bietet und wie sie in Java realisiert wird.
Vorteil : Verbesserte Typsicherheit ,
Bei Generics gibt man keinen bestimmten Datentyp an, sondern kann Klassen/Funktionen vom Typ <T t> initialisieren, wobei t einen beliebigen typen annehmen kann. Mittels Wildcards kann man sich auf bestimmte Oberklassen von Datentypen beschränken. (Numbers / Integer/ Double ..)

```
Public <T extends Number> void someMethod(){…}
Public class GenericBox <T>{
        private T item;
        public T getGenericBox{return item}
        public T setGenericBox(T item){this.item = item;}
}
```

AUFGABE 2

A) Vervollständigen sie nachfolgenden Code so, dass in der Methode print() die Werte Aller 3 x-Variablen ausgegeben werden.

```
public class outerClass{
    public int x = 0;
    class innerClass{
        public int x = 1;
        void print(int x){
            System.out.println(x);              //2
            System.out.println(this.x);         //1
            System.out.printn(outterClass.x);   //0
        }
    }
    OutterClass.innerClass ic = new OutterClass().new innerClass();
    ic.print(2);
}
```

B) Folgender Code enthält 4 Fehler, finden sie die Fehler und geben sie an wie diese korrigiert werden könnten.

```
Public class A{
    public void doSomething(){
        int i =0;
        public class A{            //innerclass kann nicht public sein
            public static void println(){  //gibt keine statischen methoden
                System.out.println("I"+ i); //muss final oder effectivly
            }
        }
        i++;        //muss final oder effectively final sein
    }}
```

C) Gegeben sei Folgende Klasse A,B und C. Instanziieren sie jede dieser Klassen in der main()-Funktion

```
Public class A{
    public class B{}
    public static class C{}
    Public static void main(String[] args){
        A a = new A();
        A.B ab = a.new B();
        A.C ac = new A.C();
    }}
```

D) Erläutern Sie, was in folgendem Code-Snippet falsch ist und wie der Fehler behoben werden könnte.

```
public void boxTest(GenericBox<Number> n){….}
public static void main(String[] args){
    boxTest(new GenericBox<Integer>());
}
public void boxTest(GenericBox<Number> n){….}
public static void main(String[] args){
    boxTest(new Integer(10));
}
```

AUFGABE 3

Gegeben seien die unten aufgeführten Klassen Client und ClientRoster zur Verwaltung eines Kundenverzeichnisses einer Firma:

A) Implementieren sie eine Methide der CleintRoster.Klasse mit folgender funktionalität:

a. public void addclient(Integer id, Client C):
Fügt einen neuen Kunden zum Kundenverzeichnis hinzu . Die Kundennummer dient als Key in der HashMap.

```
public void addClient(Integer id, Client c){
    clients.put(id,c);
}
```

b. public void removeClient(Integer customerID):
Entfernt den Kunden mit der angegebenen Kundennummer aus dem Kundenverzeichnis

```
public void removeClient(Integer id){
    clients.remove(id);
}
```

c. public ArrayList<client> filter(Predicate<client> p):
Filtert die Kundenliste nach dem gegebenen Prädikat p und gibt eine gefilterte Liste zurück. Verwenden sie zur Implementierung Keine Streams.

```
public ArrayList<Client> filter(Predicate<Client> p){
    ArrayList<Client> list = new ArrayList<>();
    Iterator<Client> iter = clients.values().iterator();

    while(iter.hasNext()){
        Client c = iter.next();
        if(p.test(c)){
            list.add(c);
        }
    }
    return list;
}
```

d. public client[] getClientsSorted(BiPredicate<Client,Client> p):
Gibt nach dem BiPredicate p sortierte Kundenliste als Array zurück. Implementieren sie zum Sortieren den BubbleSort-Algorithmus:

```
public Client[] getClientSorted(BiPredicate<Client,Client> p){
    Client[] car = new Client[clients.size()+1];
    Client temp;
    int x = 0;
    Iterator<Client> it = clients.values().iterator();
    while(it.hasNext()){
        car[x] = it.next();
        x++;
    }
    for (int i = 1; i < car.length; i++){
        for (int j = 0; j< car.length-1; j++){
            if (p.test(car[j],car[j+1])) {
                temp = car[j];
                car[j] = car[j + 1];
                car[j + 1] = temp;
            }
        }
    }
    return car;
}
```

e. public List<Client> getClientSortedAndFiltered(Comparator<Cleint> c, Predicate<Cleint>p):
Gibt eine durch das Prädikat p gefilterte und entsprechend des Comparator c sortierte Liste der Kunden zurück. implementieren sie diese Methode mit Hilfe eines parallelen Streams.

```
public List<Client> getClientSortedAndFiltered(Comparator<Client> c,
Predicate<Client> p) {
    List<Client> list = new ArrayList<Client>();
    list.addAll(clients.values());

list.parallelStream().sorted(c).filter(p).forEach(System.out::println);
}
```

B) Implementieren Sie in der unten ausgeführten Test-Klasse folgende Methodenaufrufe:

a. Filtern Sie das Kundenverzeichnis nach allen Kunden, deren Nachname mit K beginnt. Implementieren sie dazu einen Lambda-Ausdruck

```
list.stream().filter(e ->
e.lastName.startsWith("K")).forEach(System.out::println);
```

b. Filtern Sie das Kundenverzeichnis nach allen Kunden, die vor 1990 geboren wurden. Implementieren sie dies mithilfe einer anonymen Klasse

```
list.stream().filter(new Predicate<Client>() {
    public boolean test(Client client) {
        return client.yob < 1990;
    }
}).forEach(System.out::println);
```

c. Sortieren sie das Kundenverzeichnis nach Nachnamen in alphabetischer Reihenfolge. Implementieren Sie dazu einen Lambda-Ausdruck

```
list.stream().sorted((x,y) ->
x.lastName.compareTo(y.lastName)).forEach(System.out::println);
```

d. Filtern Sie das Kundenverzeichnis nach allen Kunden die vor 1990 geboren wurden und sortieren Sie diese nach Geburtsjahr.

```
list.stream().filter(x -> x.yob < 1990).sorted((x,y) ->
Integer.compare(x.yob,y.yob)).forEach(System.out::println);;
```

AUFGABE 4

A) Implementieren Sie die unten aufgeführte Methoden iterate(). Diese nimmt folgende Werte als Parameter entgegen.

- x – Startwert(double)
- n – Iterationszahl
- f: R → R – Funktion

Die Funktion f soll wiederholt (n mal) wie folgt angewendet werden: $x_{i+1} = f(x)$.
Das Ergebnis der Berechnung wird als Double-Wert zurück gegeben.

```
public Double iterate(Double x, int n, UnaryOperator<Double> f){
    while (n != 1) {
        x = f.apply(x);
        System.out.println(x);
        n--;

    }
    return x;

}
```

B) Rufen sie die Methode iterate() mit folgenden Funktionen auf:

- f(x) – 2x
- f(x) – 2^x

Implementieren Sie die Funktionen als Lambda Ausdrücke

```
public static void main(String[] args){
    ClientRoster cr = new ClientRoster();
    cr.iterate(10.0,5,(x) -> 2*x);
    cr.iterate(10.0, 5, (x)-> x*x);
}
```

AUFGABE 5

In der main Methode der folgenden Klasse wird die calculateOnShipments-Methode mittels einer anonymen Klasse aufgerufen. Implementieren Sie die zwei analoge Methoden-Aufrufe. Einmal mithilfe eines Lambda Ausdrucks und mit einer Methoden-Referenz.

```
class Shipment {
   public double calculateWeight() {
      double weight = 0;
      // Calculate weight
      return weight;
   }
public List<Double> calculateOnShipments(
   List<Shipment> l,
   Function<Shipment, Double> f){

   List<Double> results = new ArrayList<>();
   for(Shipment s : l) {
      results.add(f.apply(s));
   }
   return results;
}
public static void main(String[] args){
   List<Shipment> l = new ArrayList<Shipment>();
   l.add(new Shipment());
   Shipment shipment = new Shipment();
   // Using an anonymous class
   shipment.calculateOnShipments(l, new Function<Shipment, Double>() {
     public Double apply(Shipment s) { // The object
       return s.calculateWeight(); // The method
     }
   });
// Using a lambda expression
shipment.calculateOnShipments(l, s -> s.calculateWeight());

shipment.calculateOnShipments(l, s -> Shipment.calculateWeight(s));

// Using a method reference
shipment.calculateOnShipments(l, Shipment::calculateWeight);
```

AUFGABE 6

Gegeben sei folgende Liste:

```
List<String> month = Arrays.asList(
        "Januar", "Februar", "März","April","Mai","Juni","Juli",
        "August","September","Oktober","November","Dezember");
```

A) Implementieren Sie einen Stream, der alle Monate mit weniger als vier Buchstaben herausfiltert und die verbleibenden Monate in Großbuchstaben auf dem Bildschirm ausgibt.

```
month.stream().filter(s -> s.length() > 4).forEach(System.out::println);
```

B) Implementieren Sie einen Stream, der alle Monate mit mehr als vier Buchstaben herausfiltert und die Buchstaben der verbleibenden Monate zählt.

```
int sum = month.stream().filter(s -> s.length() <
4).map(String::length).reduce(0,(s1,s2)->s1+s2);
```

C) Implementieren Sie einen Stream der alle Monate mit weniger als vier Buchstaben herausfiltert und die verbleibenden Monate in eine Liste schreibt.

```
List<String> list = new ArrayList<>();
month.stream().filter(s -> s.length() > 4).forEach(list::add);
```

D) Implementieren Sie einen Stream der alle Monate mit weniger als vier Buchstaben herausfiltert und die verbleibenden gruppiert nach Wortlänge in eine Map schreibt.

```
month.stream().filter(s -> s.length() > 4).sorted((s1,s2) ->
s1.compareTo(s2)).forEach(System.out::println);
```

AUFGABE 7

Welche Ausgabe produzieren folgende C-Programme:

A)
```c
#include <stdio.h>

int main() {
    int a, b, c;
    a = 8;
    b = 2;
    c = 1;

    if (++b == 2 || --c && a++){
        c = 4;
    }
    printf("a: %d, b: %d, c: %d\n", a ,b ,c);
    return 0;
} a: 8, b: 3, c: 0
```

B) Gehen sie Davon aus, dass die Startadresse des arrays 0x1075bb018 ist.

```c
#include <stdio.h>
int arr[] = {10, 11, 12};
int val;
int *pi;

void p(){
    printf("arr:");
    for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); ++i)
        printf(" %d", arr[i]);
    printf(" val: %d pi: %p *pi: %d\n", val, pi, *pi);
}
int main() {
    pi = arr;

    val = *pi;      p();
    val = *pi++;    p();
    val = *++pi;    p();
    val = --*pi;    p();
    val = (*pi)++;  p();

    return 0;
}
```

C) 2

```c
int main() {
    char str[] = "str";
    printf("str: '%s' has length: %lu\n", str, sizeof(str));
    for (int i = 0; i < sizeof(str) ; ++i)
        printf("str[%d]: %c ", i, str[i]);
    printf("\n");
    return 0;
} //Output = str: 'str' has length: 4
str[0]: s str[1]: t str[2]: r str[3]: \0
```

D) Gehen Sie von einem 64-Bit System aus

```c
#include <stdio.h>

void printArray(int a[]){
    for (int i = 0; i < sizeof(a)/ sizeof(a[0]); i++)
        printf("a[%d] = %d\n", i, a[i]);
    for (int j = 0; j < sizeof(*a)/ sizeof(a[0]) ; j++) {

    }
}

int main(){
    int arr[] = {20, 21, 22, 24, 25};
    printArray(arr);
    return 0;
} //Output = a[0] = 20 // a[1] = 21 //a[0] = 20
```

E) 2

```c
#include <stdio.h>

void doSomething(int i, int arr[1], int k, int *p){
    i++; arr[0]++; k++; (*p)++;
}

int main(){
    int i = 0;
    int arr[3] = {0,0,0};
    doSomething(i, arr, arr[i], &arr[2]);
    printf("&d, %d, %d, %d", i, arr[0], arr[1], arr[2]);
    return 0;
} // Output : 0, 1, 0, 1
```

F) 2

```c
#include <stdio.h>

int a = 11, b = 22;
void swap(int a, int b){
    int tmp = a;
    a = b;
    b = tmp;
}

int main(){
    swap(a,b);
    printf("a=%d b =%d\n", a , b);
    return 0;
}//Output = a=11 b =22
```

AUFGABE 8

A) Erläutern Sie, wie aus einem C-Programmcode eine ausführbare Datei wird. Beschreiben Sie dabei, welche Aufgaben Präprozessort, Compiler und Linker haben, wie diese prinzipiell funktionieren und wie sie zusammenarbeiten. Stellen Sie auch dar, mit welchen Konsolenbefehlen Compiler und Linker aufgerufen werden (am Beispiel gcc)

AUFGABE 9

A) Definieren Sie in C einen Datentyp Person mit folgenden Eigenschaften: Name, Geburtsdatum, Geschlecht , Wohnort.

B) Vervollständigen Sie das unten aufgeführte Programm. Die Funktion findMinMax soll das Minimum und das Maximum in einem übergreifenden Array finden und Pointer darauf zurückgeben.

```c
#include <stdio.h>

int main(){
    int numbers[] = {23, 54,17,69,42};
    int *pMin, *pMax;
    int count = sizeof(numbers)/ sizeof(numbers[0]);
    findMinMax(numbers, count, &pMin,*pMax);
    printf("min: %d  max: %d\n", *pMin, *pMax);
    return 0;
}
```

C) Erläutern Sie das grundsätzliche Problem, welches bei folgendem C-Programm offensichtlich wird. Wie kann es behoben werden.

```c
#include <stdio.h>

void p( int *arr, int n){
    for (int i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}

int * returnLocalVariable(){
    int localArr[3] = {11, 22, 33};

    p(localArr, 3);
    return localArr;
}

int main(){
    int *mainArr = returnLocalVariable();

    p(mainArr,3 );
    return 0;
}
//Output 112233
```