```
//@author maximilian raspe
import java.util.Scanner;
public class Schachbrett {
public static int counter = 1;  //zählt die durchläufe, springt in neue zeile wenn fertig
public static int iterationsCounter;    //hilft um zu bestimmen, wann aus der "Schleife"
rausgesprungen werden soll
public static int switchmuster = 0;    //hilfsvariable um das schachbrettmuster zu malen

public static void main(String[] args) {

                Scanner groesse = new Scanner(System.in);
                System.out.println("Größe eingeben");
                int n = groesse.nextInt();
                groesse.close();

                iterationsCounter = n;
                System.out.println("Schachbrett : ");
                schachbrettZeichnen(n);


        }

        public static void schachbrettZeichnen(int n) {
                boolean fertig = false;//hilft um "schleife" zu beenden
                String muster1 = " X O";
                String muster2 = " O X";
                if(iterationsCounter - n >= n * n / 2) fertig = true;   //wenn n*n zeichen, also das
gesamte schachfeld entstanden ist, hört er auf zu zeichnen
                if(counter <= n / 2 && fertig == false) {       //zeichnet in zeile die felder
                        if(switchmuster % 2 == 0)System.out.print(muster1);         //switchmuster
eine gerade zahl ist, dann muster 1, ansonsten muster 2; hilft um das schachbrett muster zu
generieren
                        if(switchmuster % 2 != 0)System.out.print(muster2);
                        counter++;
                        iterationsCounter++;  //zählt die summe aller entstehenden zeichen
                        schachbrettZeichnen(n);
                }
                else if(counter > n / 2 && fertig == false) { //springt in die nächste zeile
                        counter = 1;    //setzt den zähler wieder auf 1 um n zeichen zu malen
                        System.out.println("");
                        switchmuster++;
                        schachbrettZeichnen(n);
                }
        }

}

//@author Maximiian Raspe
public class Main {

   public static void main(String[] args) {
```

```
        Bruch bruch1 = new Bruch(3, 6);
        Bruch bruch2 = new Bruch(3, 5);
        Bruch bruch3 = new Bruch(2, 3);


        Bruch result1 = Bruch.Subtrahieren(Bruch.Addieren(bruch1, bruch2), bruch3);
        System.out.println("Normaler Bruch : " + result1);
        result1.dezimalBruch();
        result1.kuerzeBruch();
        System.out.println("Gekuerzter bruch : " + result1);

        System.out.println("");

        Bruch result2 = Bruch.Dividieren(Bruch.Multiplizieren(bruch1, bruch2), bruch3);
        System.out.println("Normaler Bruch : " + result2);
        result2.dezimalBruch();
        result2.kuerzeBruch();
        System.out.println("Gekuerzter bruch : " + result2);
    }
}

import java.util.Scanner;

import static java.lang.Math.sqrt;

public class Fibonacci {

    public static long a = 0;
    public static long b = 1;
    public static int indexCounter = 1; //zählt index fi für rekursiv

    public static void main(String[] args) {

        Scanner anzahl = new Scanner(System.in);
        System.out.println("länge eingeben");
        int index = anzahl.nextInt();
        anzahl.close();


        System.out.println("Nachfolger rekursiv : " + fibonacciRekursiv(index));    //Zeit: 0.02068845
ms für anzahl 5
        System.out.flush();
        System.out.println("Nachfolger iterativ : " + fibonacciIterativ(index));    //4.7013E-4 ms für
anzahl 5
        System.out.flush();
        System.out.println("Fibonacci formel : " +fibonacciFormel(index));  // 2.73465E-4 ms für
anzahl 5
        System.out.flush();
        /* 1.5
        Vorteile Rekursiv: -einfacher zu verstehen und sieht besser aus; ist einfacher zu programmieren
als iterativ;
        Nachteile Rekursiv:- wird mit steigender zahl immer speicherintensiver
```

```
      Vorteile Iterativ: -eine optimierte iterative lösung ist schneller als rekursiv mit steigender zahl
      Nachteile Iterativ: -ist schwerer zu verstehen bzw unleserlicher

      Vorteile Formel:-liefert schnell ergebnisse mit geringem speicher verbrauch
      Nachteile Formel:- es kommen rundungsfehler zustande
       */
   }

   public static long fibonacciRekursiv(int index) {
      long temp;   //hilfsvariable
      if(index > indexCounter){  // läuft solange bis eingegebner index > indexCounter ist
         temp = a + b;
         b = a;
         a = temp;
         indexCounter++;
         fibonacciRekursiv(index);
      }
      return a + b;   //gibt das glied an i-ter stelle aus

   }

   public static long fibonacciIterativ(int index) {
      long a = 0;
      long b = 1;
      long temp;

      for(int i = 1; i < index; i++) {    //bricht die schleife ab wenn entsprechender index erreicht
wurde
         temp = a + b;
         b = a;
         a = temp;
      }
      return a + b;   //gibt folgeglied aus
   }

   public static long fibonacciFormel(int index) {
      double ergebnis = (1 / sqrt(5)) * ((Math.pow(((1 + sqrt(5)) / 2), index)) - (Math.pow(((1 -
sqrt(5)) / 2), index)));    //berechnet fibonacci zahl für eingegeben index
      return (long)ergebnis;
   }
}

package com.company;
import java.util.Scanner;

public class Euklid {

public static int a;
public static int b;

   public static void main(String[] args) {
```

```
        Scanner zahlen = new Scanner(System.in);
        System.out.println("Bitte Zahl a eingeben");
        a = zahlen.nextInt();
        System.out.println("Bitte Zahl b eingeben");
        b = zahlen.nextInt();
        zahlen.close();

        ggT(a,b);
    }
    public static void ggT(int a, int b) {
        if (a == b) {   // wenn a = b, dann wird der ggT ausgegeben
            System.out.println("ggT : " + a);
            System.exit(0);
        }
        if(a > b) { //wenn a > b, dann wird subtrahiert und mit den neuen werten ggT aufgerufen
            a = a - b;
            ggT(a,b);
        }
        if(a < b) { //wenn b > a, dann wird subtrahiert und mit den neuen werten ggT aufgerufen
            b = b - a;
            ggT(a,b);
        }
    }
}

//@author Maximiian Raspe
public class Bruch {
    //4.1
    private int zaehler;
    private int nenner;


    public int getNenner() {    //getter method
        return this.nenner;
    }

    public int getZaehler() {   //getter method
        return this.zaehler;
    }
    //4.2
    public Bruch(int zaehler, int nenner) { //konstruktor

        if (nenner == 0) {
            System.out.println("Nenner ungleich 0 bitte");
            System.exit(-1);    //beendet programm mit fehlermeldung
        } else {
            this.zaehler = zaehler;
            this.nenner = nenner;
        }

    }
```

```
    public Bruch(int ganzzahl) {    //bruch für ganzzahl

        this(ganzzahl, 1);
    }

    public String toString() {  //@override toString
        return this.zaehler + " / " + this.nenner;
    }

    //4.4   //grundrechenarten für brüche
    public static Bruch Multiplizieren(Bruch bruch1, Bruch bruch2) {
        int zaehler = bruch1.getZaehler() * bruch2.getZaehler();
        int nenner = bruch1.getNenner() * bruch2.getNenner();
        return new Bruch(zaehler, nenner);
    }
    public static Bruch Dividieren(Bruch bruch1, Bruch bruch2) {
        int zaehler = bruch1.getZaehler() * bruch2.getNenner();
        int nenner = bruch1.getNenner() * bruch2.getZaehler();
        return new Bruch(zaehler, nenner);
    }
    public static Bruch Addieren(Bruch bruch1, Bruch bruch2) {
        int zaehler = bruch1.getZaehler() * bruch2.getNenner() + bruch2.getZaehler() *
bruch1.getNenner();
        int nenner = bruch1.getNenner() * bruch2.getNenner();
        return new Bruch(zaehler, nenner);
    }
    public static Bruch Subtrahieren(Bruch bruch1, Bruch bruch2) {
        int zaehler = bruch1.getZaehler() * bruch2.getNenner() - bruch2.getZaehler() *
bruch1.getNenner();
        int nenner = bruch1.getNenner() * bruch2.getNenner();
        return new Bruch(zaehler, nenner);
    }

    //4.5
    public void kuerzeBruch() {

        zaehler = getZaehler();
        nenner = getNenner();

        for(int i = 1; i <= zaehler && i <= nenner; i++) {  //findet den größten teiler und kürzt
dementsprechened den bruch
            if(zaehler % i == 0 && nenner % i == 0) {
                zaehler = zaehler / i;
                nenner = nenner / i;
            }
        }
        new Bruch(zaehler, nenner); //setzt den neuen gekürzten bruch
    }
    public void dezimalBruch() {
        int zaehler = getZaehler();
        int nenner = getNenner();
```

```
        double dezimal = (double)zaehler / (double)nenner;  //berechnet den dezimal wert des bruchs
        System.out.println("Dezimalwert : " + dezimal);
    }
}
```