Freie und Hansestadt Hamburg
Behörde für Schule und Berufsbildung

_____/_____
Name / Kurs-Nr.

Schriftliche Abiturprüfung
Schuljahr 2024/2025

Informatik

auf grundlegendem Anforderungsniveau

an allgemeinbildenden und beruflichen gymnasialen Oberstufen

Haupttermin
Freitag, 16. Mai 2025, 09:00 Uhr

Unterlagen für die Prüflinge

Allgemeine Arbeitshinweise

- Überprüfen Sie diese Unterlagen auf Vollständigkeit.
- Schreiben Sie auf alle Prüfungsunterlagen Ihren Namen und zusätzlich auf dieses Deckblatt Ihre Kursnummer.
- Kennzeichnen Sie Ihre Entwurfsblätter (Kladde) und Ihre Reinschrift.

Fachspezifische Arbeitshinweise[1]

- Die Arbeitszeit beträgt **255 Minuten**.
- Eine gesonderte Lese- oder Auswahlzeit wird nicht gewährt.
- Hilfsmittel: Taschenrechner (nicht programmierbar, nicht grafikfähig), Rechtschreibwörterbuch.

Aufgabenauswahl

- Sie erhalten **drei** Aufgaben zu unterschiedlichen Schwerpunkten.
- Bearbeiten Sie die **Pflichtaufgabe (Aufgabe I)** und **genau eine** weitere Aufgabe (Aufgabe II oder III).
- Wählen Sie jeweils eine der angegebenen Programmiersprachen.
- Vermerken Sie auch auf Ihrer Reinschrift, welche Aufgaben Sie ausgewählt und bearbeitet haben.

Bearbeitet wurden die folgenden Aufgaben (bitte kreuzen Sie an):

I	Objektorientierte Modellierung und Programmierung (Pflicht)	(☐ Java ☐ Python)
II	Datensicherheit in verteilten Systemen	(☐ Java ☐ Python)
III	Intelligente Suchverfahren	(☐ Java ☐ Python)

[1] Entsprechend der „Richtlinie über die Gewährung von Erleichterungen für neu zugewanderte Schülerinnen, Schüler und Prüflinge bei Sprachschwierigkeiten in der deutschen Sprache“ (MBlSchul Nr. 08, 7. Oktober 2016, S. 60) werden für die betroffenen Prüflinge die folgenden Erleichterungen gewährt:
- Die Bearbeitungszeit wird um 30 Minuten auf **285 Minuten** erhöht.
- Ein nicht-elektronisches Wörterbuch Deutsch – Herkunftssprache / Herkunftssprache – Deutsch wird bereitgestellt.

Bewertung

Jeder Aufgabe sind 50 Bewertungseinheiten (BE) zugeordnet. In allen Teilaufgaben werden nur ganze oder halbe BE vergeben. Insgesamt sind 100 BE erreichbar. Bei der Festlegung von Notenpunkten gilt die folgende Tabelle.

Erbrachte Leistung (in BE)	Notenpunkte
≥ 95	15
≥ 90	14
≥ 85	13
≥ 80	12
≥ 75	11
≥ 70	10
≥ 65	9
≥ 60	8

Erbrachte Leistung (in BE)	Notenpunkte
≥ 55	7
≥ 50	6
≥ 45	5
≥ 40	4
≥ 33	3
≥ 27	2
≥ 20	1
< 20	0

Für die Erteilung der **Note gut** (11 Punkte) ist mindestens erforderlich, dass mindestens 75 % der erwarteten Gesamtleistung erbracht werden. Dabei muss die Prüfungsleistung in ihrer Gliederung, in der Gedankenführung, in der Anwendung fachmethodischer Verfahren sowie in der fachsprachlichen Artikulation den Anforderungen voll entsprechen.

Für die Erteilung der **Note ausreichend** (5 Punkte) ist mindestens erforderlich, dass mindestens 45 % der erwarteten Gesamtleistung und über den Anforderungsbereich I hinaus Leistungen in einem weiteren Anforderungsbereich erbracht werden.

Die zwei voneinander unabhängigen Aufgaben der Prüfungsaufgabe werden jeweils mit 50 Bewertungseinheiten bewertet. Die erbrachte Gesamtleistung ergibt sich aus der Summe der Bewertungseinheiten in den beiden Aufgaben.

Schwerwiegende und gehäufte Verstöße gegen die sprachliche Richtigkeit oder gegen die äußere Form führen zu einem Abzug von bis zu zwei Punkten in einfacher Wertung. Mangelhafte Gliederung, Fehler in der Fachsprache, Ungenauigkeiten in Zeichnungen oder unzureichende oder falsche Bezüge zwischen Zeichnungen und Text sind als fachliche Fehler zu werten. Ein Abzug für Verstöße gegen die sprachliche Richtigkeit soll nicht erfolgen, wenn diese bereits Gegenstand der fachspezifischen Bewertungsvorgaben sind.

Aufgabe I: Campingplatzplan (Java-Version) 50 BE

Schwerpunkt: Objektorientierte Modellierung und Programmierung von Grafiksystemen

Im Folgenden soll ein Platzplan für einen Campingplatz entwickelt werden.

Abbildung 1: Campingplatz
Quelle: Dietmar Rabich / Wikimedia Commons / „Haltern am See, Campingplatz -Drügen Kamp- -- 2014 -- 9036“ / CC BY-SA 4.0

Mit dem Plan bekommen Besucherinnen und Besucher eine Übersicht über den Platzaufbau und können eine Auswahl eines von ihnen gewünschten Stellplatzes für Wohnmobil, Wohnwagen oder Zelt durchführen.

Zu einem ersten Entwurf sind hier das Bild eines Teils des Platzplans (Abbildung 2) und ein Klassendiagramm (Abbildung 3) angegeben:

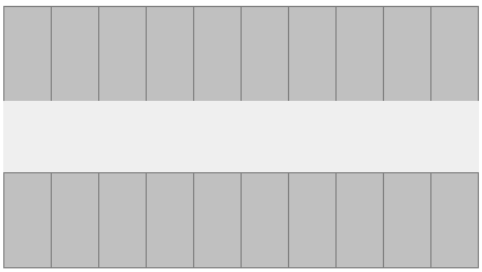


Abbildung 2: Teil des Platzplans

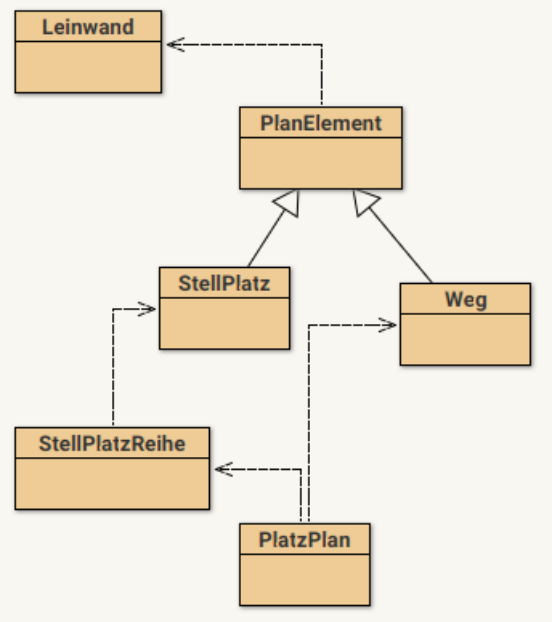


Abbildung 3: Klassendiagramm

Hinweis: Vereinfachte Klassenkarten von `PlanElement`, `StellPlatz` *und* `StellPlatzReihe` *finden Sie in der Anlage 1.*

I.a
- **Geben** Sie **an**, welche Objekte für das Bild des Platzplans (siehe Abbildung 2) durch die Klasse `PlatzPlan` erzeugt werden müssen.
- **Erläutern** Sie den Unterschied zwischen Klassen und Objekten an einem Beispiel im Kontext Campingplatzplan.

(10 BE)

I.b
- **Erläutern** Sie die Beziehungen, die aus dem Klassendiagramm erkennbar sind.
- **Geben** Sie **an**, woran die Beziehungen jeweils im Programmtext erkennbar sind.

(10 BE)

Die im Klassendiagramm angegebene Klasse `PlanElement` ist im Klassendiagramm nicht als abstrakte Klasse modelliert worden. Sie sollte aber abstrakt sein.

I.c **Erläutern** Sie, weshalb die Klasse `PlanElement` abstrakt sein sollte.

(6 BE)

Stellplätze und Wege sollen beschriftet werden können und die Stellplatzobjekte sollen ihren Belegungszustand halten. Die Beschriftung kann einfach eine Nummer sein, sie kann aber auch andere Zeichen enthalten, die beispielsweise verschiedene Platzbereiche kennzeichnen.

I.d **Untersuchen** Sie, welche notwendigen Attribute und Methoden von der Klasse `StellPlatz` (oder alternativ `PlanElement`) bereitgestellt werden müssen, damit von der Campingplatzverwaltung das Stellplatzobjekt mit der gesuchten Beschriftung gefunden und damit außerdem vom Stellplatzobjekt abgefragt werden kann, ob es aktuell belegt ist.

(6 BE)

Auf Campingplätzen gibt es in der Regel verschiedene Typen von Stellplätzen, die sich beispielsweise in ihrer Ausstattung unterscheiden. So haben Standardplätze manchmal keinen eigenen Stromanschluss, Komfortplätze dafür neben einem eigenen Strom- auch einen eigenen Wasser- sowie einen eigenen Abwasseranschluss. Den Quelltext zur Klasse `StellPlatz`, die diese Ausstattungsmerkmale berücksichtigt, finden Sie in der Anlage 2.

I.e **Analysieren** Sie den Programmtext der angegebenen Klasse `StellPlatz` (Anlage 2) dahingehend, an welchen Stellen und zu welchem Zweck die oben genannten Ausstattungsmerkmale der Plätze berücksichtigt werden.

(6 BE)

Die Darstellung von Hecken zwischen den einzelnen Stellplätzen ist für die Kunden interessant, weil der Aufenthalt durch den größeren Abstand zu den Nachbarn und dem sich dadurch ergebenden Sichtschutz unter Umständen angenehmer wird. Eine Hecke besteht hierbei aus mehreren Büschen (siehe dazu das Bild in der Anlage 3).

I.f **Stellen** Sie **dar**, welche Auswirkungen der Satz *„Eine Hecke besteht aus mehreren Büschen“* auf Ihre Modellierung hinsichtlich einer Erweiterung des Projekts um die Klassen Hecke und Busch hat.

(6 BE)

In den bisherigen Überlegungen wurde davon ausgegangen, dass alle Stellplätze und Wege eine einfache rechteckige Form haben. Der Rumpf der Klassendefinition von `StellPlatz` ist im Folgenden in einer für diese Teilaufgabe vereinfachten Version dargestellt und besteht nur aus dem Konstruktor und einer einzigen Methode:

```
1  /**
2   * Erzeuge einen neuen StellPlatz
3   */
4  public StellPlatz(int x, int y, int b, int t, int w)
5  {
6      super(x,y,b,t,w,"schwarz","hellgrau",true);
7  }
8
9  /**
10  * Berechnet das zu zeichnende Shape anhand der gegebenen Daten
11  */
12 protected Shape gibFigur()
13 {
14     int breite=gibBreite();
15     int tiefe=gibTiefe();
16     Shape stellPlatz = new Rectangle2D.Double(0,0,breite,tiefe);
17
18     return  transformiere(stellPlatz);
19 }
```

Es soll die Möglichkeit für andere Formen geschaffen werden. Dazu ist in der Klasse `StellPlatz` ein Attribut

```
private ArrayList<Punkt> punktListe
```

vorgesehen, dessen Punkte durch

```
new Punkt(0,0);
new Punkt(breite,0);
new Punkt(breite,tiefe);
new Punkt(0,tiefe);
```

für eine Methode erzeugt werden können, die zur oben angegebenen für rein rechteckige Formen passt.
(Den Programmtext zur Klasse `Punkt` finden Sie in der Anlage 4.)

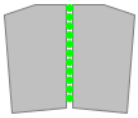


Abbildung 4: Beispiel für eine nicht rechteckige Form

I.g **Untersuchen** Sie die in der Anlage 5 angegebene Methode `gibFigur()`, mit der auch andere als rechteckige Stellplätze dargestellt werden können. Sie verwendet dabei zu einem vorher erzeugten GeneralPath-Objekt `path` die Methodenaufrufe für einen „Zeichenstift“:

`GeneralPath path = new GeneralPath();`
`path.moveTo( <xKoordinate> , <yKoordinate> );` (setzt den Zeichenstift auf die übergebene Position)
`path.lineTo( <xKoordinate> , <yKoordinate> );` (zeichnet eine Linie von der jeweiligen Position zum übergebenen Punkt)

Hinweis: GeneralPath-Objekte implementieren (ebenso wie Rectangle2D-Objekte) die Schnittstelle Shape, dürfen also überall dort auftauchen, wo ein Shape verlangt wird, wie es bei `gibFigur()` *der Fall ist.*

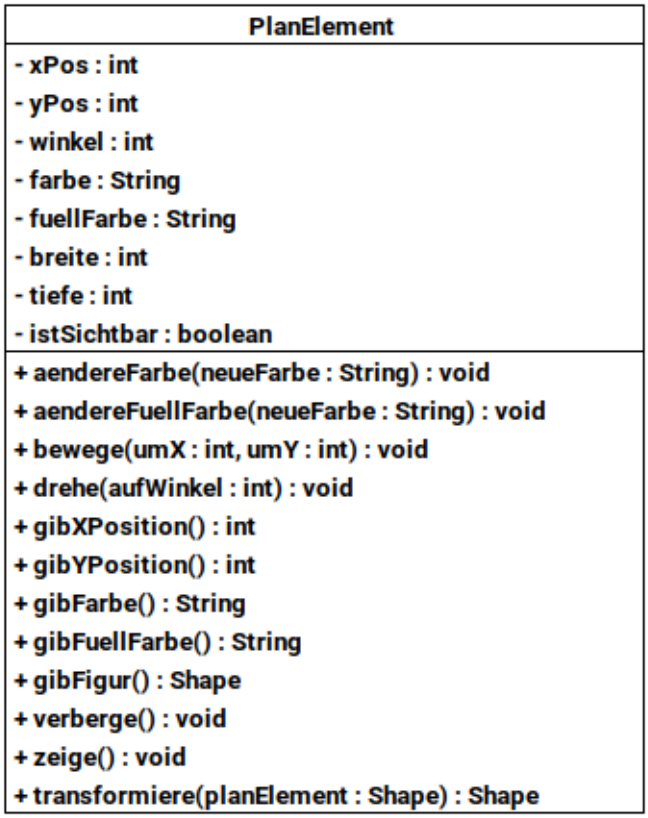
(6 BE)

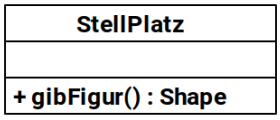
Anlage zur Aufgabe I: Campingplatzplan (Java-Version)

Anlage 1: Klassenkarten *(vereinfachte Darstellung)*

Hinweis: Die Konstruktormethoden wurden jeweils wegen der hohen Parameteranzahl weggelassen. Für PlanElement lautet der Konstruktor zum Beispiel wie folgt:

```
+PlanElement(x: int, y: int, b: int, t: int, w: int, f: String, ff: String,
s: boolean)
```





Die Klassenkarte für die `StellPlatzReihe` *wurde weiter vereinfacht und wird ohne Methoden dargestellt.*

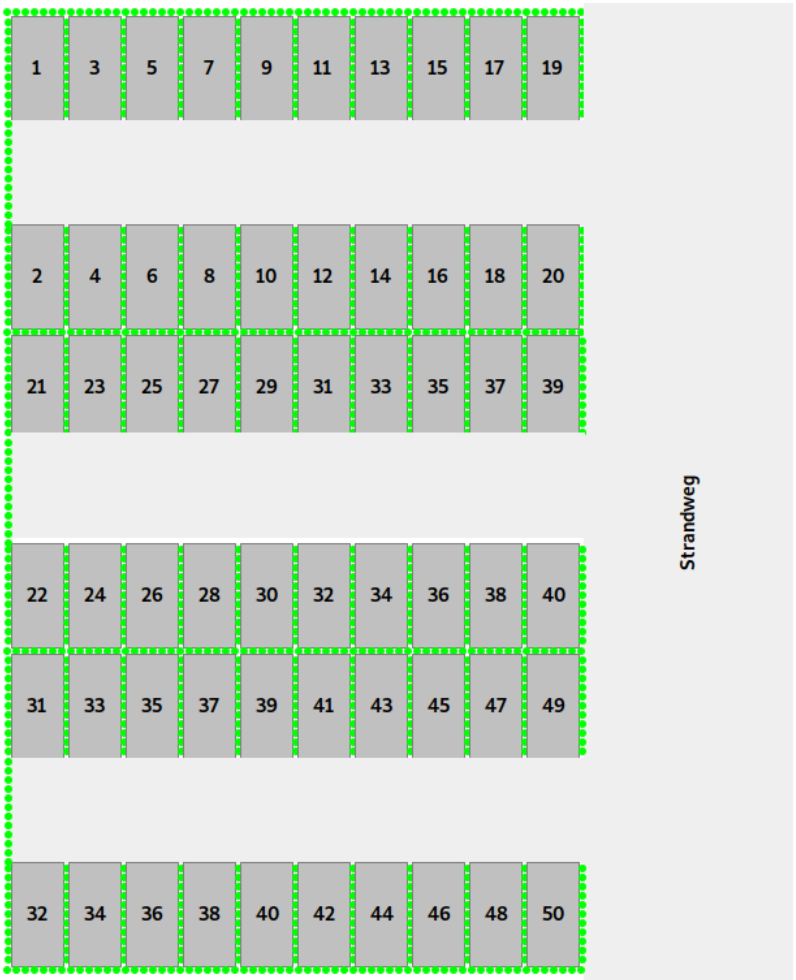
Anlage 2: Quelltext der Klasse `StellPlatz`

```
1  public class StellPlatz
2  extends PlanElement
3  {
4      private String ausstattung="Standard";
5
6      /**
7       * Erzeuge einen neuen StellPlatz mit Ausstattung
8       */
9      public StellPlatz(int x, int y, int b, int t, int w, String ausstattung) {
10         super(x,y,b,t,w,"schwarz","hellgrau",true);
11         aendereAusstattung(ausstattung);
12     }
13
14     /**
15      * Berechnet das zu zeichnende Shape anhand der gegebenen Daten
16      */
17     protected Shape gibFigur()
18     {
19         int breite=gibBreite();
20         int tiefe=gibTiefe();
21         Shape stellPlatz = new Rectangle2D.Double(0,0,breite,tiefe);
22
23         return  transformiere(stellPlatz);
24     }
25
26     /**
27      * gibt abhaengig von der Ausstattung eine Farbe fuer die Fuellung zurueck
28      */
29     public String farbeZuAusstattung(String ausstattung) {
30         if (ausstattung=="Komfort") return "gelb";
31         if (ausstattung=="Zelt") return "hellgruen";
32         return "hellgrau";
33     }
34
35     /**
36      * aendert die Art der Ausstattung
37      */
38     public void aendereAusstattung(String ausstattung) {
39         this.ausstattung=ausstattung;
40         aendereFuellFarbe(farbeZuAusstattung(ausstattung));
41     }
42 }
```

Hinweis: Die Klasse `PlanElement` *enthält die folgende (hier vereinfachte) Methode* `aendereFuellfarbe`*:*

```
public void aendereFuellFarbe(String neueFarbe) {
    verberge();
    fuellFarbe = neueFarbe;
    zeige();
}
```

Anlage 3: Platzplan mit Hecken



Anlage 4: Die Klasse `Punkt`

```
1  public class Punkt
2  {
3      private int x,y;
4
5      public Punkt(int x, int y)
6      {
7          this.x = x;
8          this.y = y;
9      }
10
11     public int gibX() { return x; }
12     public int gibY() { return y; }
13 }
```

Anlage 5: Die Methode `gibFigur()`

```
1  /**
2   * Berechnet das zu zeichnende Shape anhand der gegebenen Daten
3   */
4  protected Shape gibFigur()
5  {
6      GeneralPath stellPlatz = new GeneralPath();
7      stellPlatz.moveTo(punktListe.get(0).gibX(),
8                                  punktListe.get(0).gibY());
9      for (int i=1; i<punktListe.size(); i++) {
10         stellPlatz.lineTo(punktListe.get(i).gibX(),
11                                     punktListe.get(i).gibY());
12         }
13     stellPlatz.lineTo(punktListe.get(0).gibX(),
14                       punktListe.get(0).gibY());
15
16     return  transformiere(stellPlatz);
17 }
```

Aufgabe I: Campingplatzplan (Python-Version) 50 BE

Schwerpunkt: Objektorientierte Modellierung und Programmierung von Grafiksystemen

Im Folgenden soll ein Platzplan für einen Campingplatz entwickelt werden.

Abbildung 1: Campingplatz
Quelle: Dietmar Rabich / Wikimedia Commons / „Haltern am See, Campingplatz -Drügen Kamp- -- 2014 -- 9036“ / CC BY-SA 4.0

Mit dem Plan bekommen Besucherinnen und Besucher eine Übersicht über den Platzaufbau und können eine Auswahl eines von ihnen gewünschten Stellplatzes für Wohnmobil, Wohnwagen oder Zelt durchführen.

Zu einem ersten Entwurf sind hier das Bild eines Teils des Platzplans (Abbildung 2) und ein Klassendiagramm (Abbildung 3) angegeben:

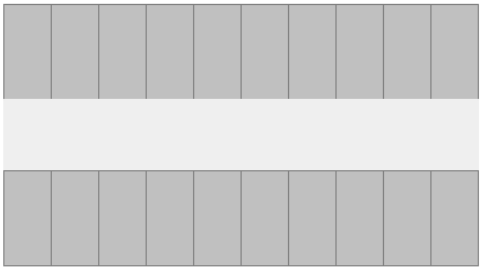


Abbildung 2: Teil des Platzplans

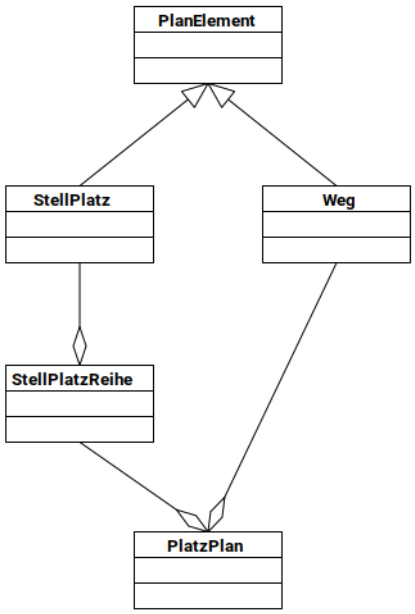


Abbildung 3: Klassendiagramm

Hinweis: Die Klassen `Grafikfenster` *und* `Zeichenflaeche` *wurden weggelassen. Vereinfachte Klassenkarten von* `PlanElement`, `StellPlatz` *und* `StellPlatzReihe` *finden Sie in der Anlage 1.*

I.a
- **Geben** Sie **an**, welche Objekte für das Bild des Platzplans (siehe Abbildung 2) durch die Klasse `PlatzPlan` erzeugt werden müssen.
- **Erläutern** Sie den Unterschied zwischen Klassen und Objekten an einem Beispiel im Kontext Campingplatzplan.

(10 BE)

I.b
- **Erläutern** Sie die Beziehungen, die aus dem Klassendiagramm erkennbar sind.
- **Geben** Sie **an**, woran die Beziehungen jeweils im Programmtext erkennbar sind.

(10 BE)

Die im Klassendiagramm angegebene Klasse `PlanElement` ist im Klassendiagramm nicht als abstrakte Klasse modelliert worden. Sie sollte aber abstrakt sein.

I.c **Erläutern** Sie, weshalb die Klasse `PlanElement` abstrakt sein sollte.

(6 BE)

Stellplätze und Wege sollen beschriftet werden können und die Stellplatzobjekte sollen ihren Belegungszustand halten. Die Beschriftung kann einfach eine Nummer sein, sie kann aber auch andere Zeichen enthalten, die beispielsweise verschiedene Platzbereiche kennzeichnen.

I.d **Untersuchen** Sie, welche notwendigen Attribute und Methoden von der Klasse `StellPlatz` (oder alternativ `PlanElement`) bereitgestellt werden müssen, damit von der Campingplatzverwaltung das Stellplatzobjekt mit der gesuchten Beschriftung gefunden und damit außerdem vom Stellplatzobjekt abgefragt werden kann, ob es aktuell belegt ist.

(6 BE)

Auf Campingplätzen gibt es in der Regel verschiedene Typen von Stellplätzen, die sich beispielsweise in ihrer Ausstattung unterscheiden. So haben Standardplätze manchmal keinen eigenen Stromanschluss, Komfortplätze dafür neben einem eigenen Strom- auch einen eigenen Wasser- sowie einen eigenen Abwasseranschluss. Den Quelltext zur Klasse `StellPlatz`, die diese Ausstattungsmerkmale berücksichtigt, finden Sie in der Anlage 2.

I.e **Analysieren** Sie den Programmtext der angegebenen Klasse `StellPlatz` (Anlage 2) dahingehend, an welchen Stellen und zu welchem Zweck die oben genannten Ausstattungsmerkmale der Plätze berücksichtigt werden.

(6 BE)

Die Darstellung von Hecken zwischen den einzelnen Stellplätzen ist für die Kunden interessant, weil der Aufenthalt durch den größeren Abstand zu den Nachbarn und dem sich dadurch ergebenden Sichtschutz unter Umständen angenehmer wird. Eine Hecke besteht hierbei aus mehreren Büschen (siehe dazu das Bild in der Anlage 3).

I.f **Stellen** Sie **dar**, welche Auswirkungen der Satz *„Eine Hecke besteht aus mehreren Büschen“* auf Ihre Modellierung hinsichtlich einer Erweiterung des Projekts um die Klassen `Hecke` und `Busch` hat.

(6 BE)

In den bisherigen Überlegungen wurde davon ausgegangen, dass alle Stellplätze und Wege eine einfache rechteckige Form haben. Der Rumpf der Klassendefinition von `StellPlatz` ist im Folgenden in einer für diese Teilaufgabe vereinfachten Version dargestellt und besteht nur aus dem Konstruktor und einer einzigen Methode:

```
1  def __init__(self,
2                xPos=60,
3                yPos=20,
4                breite=40,
5                tiefe=80,
6                winkel=0,
7                farbe='GRAY',
8                fuellfarbe="LIGHT GRAY"):
9      PlanElement.__init__(self, xPos, yPos, breite, tiefe, winkel,
10                          farbe, fuellfarbe)
11
12 def GibFigur(self):
13     """definiert die zu zeichnende Figur"""
14     path = Zeichenflaeche.GibZeichenflaeche().GibGC().CreatePath()
15
16     # lokale Variable zur Schreibvereinfachung
17     b, t = self.GibBreite(), self.GibTiefe()
18     path.AddRectangle(0, 0, b, t)
19
20     return self.Transformiere(path)
```

Es soll die Möglichkeit für andere Formen geschaffen werden. Dazu wird im Konstruktor der Klasse `StellPlatz` ein Attribut

`self.punktListe`

eingeführt, dessen Werte für eine rechteckige Form durch

`[(0,0),(breite,0),(breite,tiefe),(0,tiefe)]`

gegeben sind.

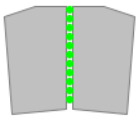


Abbildung 4: Beispiel für eine nicht rechteckige Form

I.g **Untersuchen** Sie die in der Anlage 4 angegebene Methode `GibFigur`, mit der auch andere als rechteckige Stellplätze dargestellt werden können. Sie verwendet dabei die Methodenaufrufe für einen „Zeichenstift“

`path.MoveToPoint( <xKoordinate> , <yKoordinate> ),`

die ihn auf die übergebene Position setzt und

`path.AddLineToPoint( <xKoordinate> , <yKoordinate> ),`

die von der jeweiligen Position eine Linie zum übergebenen Punkt zeichnet.

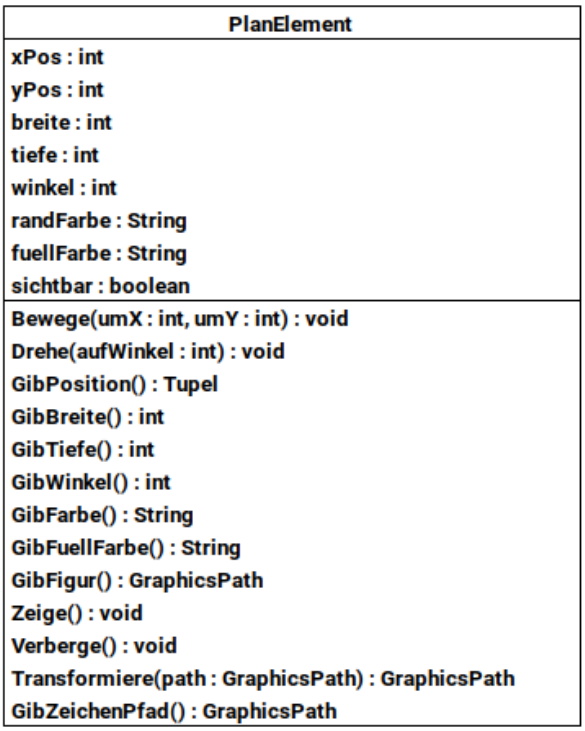
(6 BE)

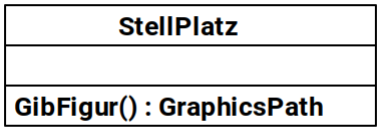
Anlage zur Aufgabe I: Campingplatzplan (Python-Version)

Anlage 1: Klassenkarten *(vereinfachte Darstellung)*

Hinweis: Die Konstruktormethoden wurden jeweils wegen der hohen Parameteranzahl weggelassen. Für PlanElement lautet der Konstruktor zum Beispiel wie folgt:

```
__init__(self, xPos:int, yPos:int, breite:int, tiefe:int, winkel:int, farbe:String, fuellfarbe:String):PlanElement
```





Die Klassenkarte für die `StellPlatzReihe` *wurde weiter vereinfacht und wird ohne Methoden dargestellt.*

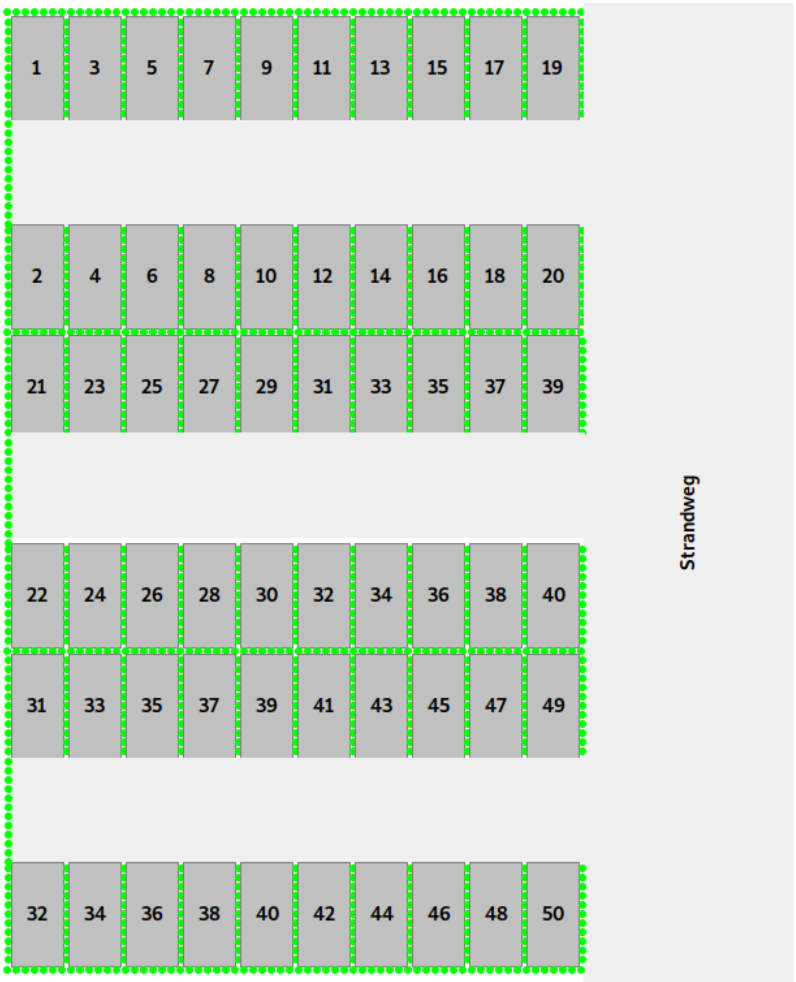
Anlage 2: Quelltext der Klasse `StellPlatz`

```
1  class StellPlatz(PlanElement):
2
3      def __init__(self,
4                   xPos=60,
5                   yPos=20,
6                   breite=40,
7                   tiefe=80,
8                   winkel=0,
9                   farbe='BLACK',
10                  ausstattung='Standard'):
11         self.ausstattung=ausstattung
12         fuellfarbe=self.FarbeZuAusstattung(ausstattung)
13         PlanElement.__init__(self, xPos, yPos, breite, tiefe, winkel,
14                              farbe, fuellfarbe)
15
16     def GibFigur(self):
17         """definiert die zu zeichnende Figur"""
18         path = Zeichenflaeche.GibZeichenflaeche().GibGC().CreatePath()
19
20         # lokale Variable zur Schreibvereinfachung
21         b, t = self.GibBreite(), self.GibTiefe()
22         path.AddRectangle(0, 0, b, t)
23
24         return self.Transformiere(path)
25
26     def FarbeZuAusstattung(self, ausstattung):
27         if ausstattung=='Komfort':
28             return "YELLOW"
29         if ausstattung=='Zelt':
30             return "LIME GREEN"
31         return "LIGHT GRAY"
32
33     def AendereAusstattung(self, ausstattung):
34         self.ausstattung=ausstattung
35         self.AendereFuellFarbe(self.FarbeZuAusstattung(ausstattung))
```

Hinweis: Die Klasse `PlanElement` *enthält die folgende (hier vereinfachte) Methode* `AendereFuellfarbe`*:*

```
    def AendereFuellFarbe(self, neueFuellFarbe):
        """Veraendernde Methode fuer die Fuellfarbe"""
        self.Verberge()
        self.__ff = neueFuellFarbe
        self.Zeige()
```

Anlage 3: Platzplan mit Hecken



Anlage 4: Quelltext der Methode `GibFigur`

```
1  def GibFigur(self):
2      """definiert die zu zeichnende Figur"""
3      path = self.GibZeichenPfad()
4
5      x0,y0=self.punktListe[0]
6      path.MoveToPoint(x0,y0)
7      for punkt in self.punktListe[1:]:
8          x,y=punkt
9          path.AddLineToPoint(x,y)
10     path.AddLineToPoint(x0,y0)
11
12     return self.Transformiere(path)
```

Aufgabe II: Virtual Private Network (Java-Version) 50 BE

Schwerpunkt: Datensicherheit in verteilten Systemen

Eva möchte von unterwegs auf ihre Daten zugreifen können, die sie zu Hause auf ihrem Server gespeichert hat. Dazu möchte sie eine Internetverbindung nutzen. Da sie auf ihrem Server auch vertrauliche Daten hat, möchte sie von ihrem Laptop aus, welchen sie auf ihren Reisen immer dabei hat, eine sogenannte VPN-Verbindung (VPN steht hier für „Virtual Private Network“; es handelt sich also in der Regel um eine nach außen hin verborgene und verschlüsselte Verbindung) zu ihrem Server zu Hause herstellen können. Folgenden Aufbau hat sie sich überlegt (Abbildung 1):

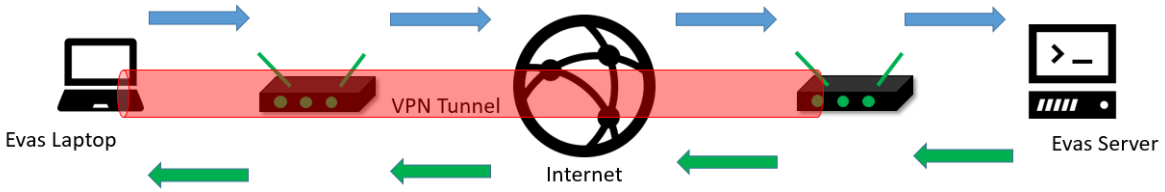


Abbildung 01

II.a **Stellen** Sie die Relevanz der drei folgenden Kriterien beim Einsatz einer verschlüsselten VPN-Verbindung **dar**:
- Authentizität
- Integrität
- Vertraulichkeit.

(10 BE)

Stellt Eva mit ihrem Laptop eine Verbindung zu ihrem Server zu Hause her, baut sie eine VPN-Verbindung im sogenannten Tunnelmodus zu ihrem Server auf. In vereinfachter Form sind die zu übertragenden Pakete inkl. der relevanten Header folgendermaßen aufgebaut (Abbildung 2):

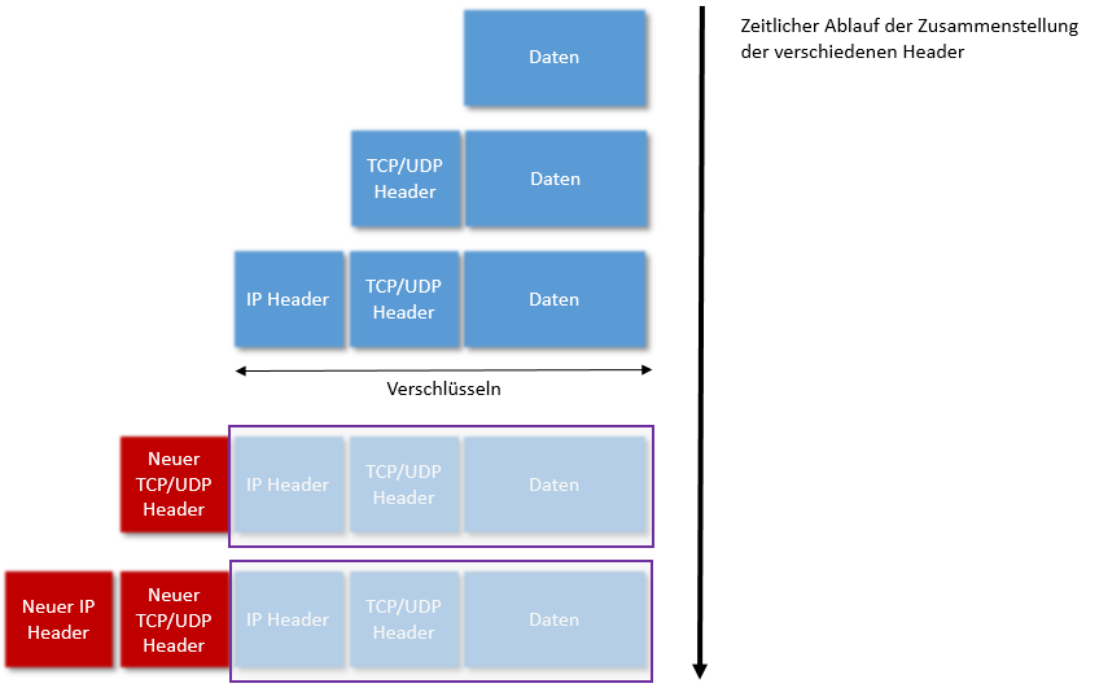


Abbildung 2

II.b
- **Beschreiben** Sie anhand der Abbildung 2, wie ein VPN-Tunnel entsteht und was man sich unter einem VPN-Tunnel vorzustellen hat.
- **Stellen** Sie **dar**, warum es zusätzlich zu dem Aufbau eines Tunnels nötig ist, die zu übermittelnden Daten zu verschlüsseln.

(11 BE)

Der VPN-Anbieter *Wireguard* nutzt den symmetrischen Verschlüsselungsalgorithmus ChaCha20. Er ist im Vergleich zu anderen modernen symmetrischen Verfahren (wie z. B. AES) sehr einfach, aber dennoch sehr sicher. Er funktioniert wie folgt:
Zunächst wird aus einem (256 Bit langen) symmetrischen Schlüssel ein sogenannter „Streaming-Schlüssel“ gebildet, der an die Länge des Klartextes angepasst wird. Anschließend wird der (binärcodierte) Klartext über die XOR-Verknüpfung mit dem „Streaming-Schlüssel“ verknüpft. (Für die XOR-Verknüpfung siehe Anlage 1).
Eva zweifelt das Verfahren an. Sie meint, dass die XOR-Verknüpfung nicht eindeutig wieder zu entschlüsseln ist.

II.c **Untersuchen** Sie, ob Evas Zweifel gerechtfertigt sind und die Entschlüsselung bei der XOR-Verknüpfung tatsächlich nicht eindeutig ist. Wie die XOR-Verknüpfung die Bits verarbeitet, lesen Sie in der Anlage 1.

(8 BE)

Eva hat begonnen, in einem kleinen Programm die ChaCha20-Verschlüsselung zu implementieren. Zunächst können nur Großbuchstaben verschlüsselt werden, z. B.:
Mit dem Klartext „HAL“
und dem folgenden Streaming-Schlüssel in passender Länge

```
[0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1]
```

würde sich die Chiffre

```
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]
```

ergeben.
Evas Programmcode finden Sie in der Anlage 2.

II.d **Erklären** Sie die Funktionsweise des Programms im Detail (zur Verarbeitung der Bits mit der XOR-Verknüpfung siehe Anlage 1).

(10 BE)

Der geheime, symmetrische Sitzungs-Schlüssel, den ChaCha20 verwendet, ist 256 Bit lang. Dieser Schlüssel wird für jede VPN-Sitzung neu berechnet und muss anschließend an beiden Tunnelenden bekannt bzw. vorhanden sein. Anschließend wird aus diesem Sitzungsschlüssel durch die Zugabe verschiedener Operationen ein pseudozufälliger Streaming-Schlüssel, der der Länge des zu verschlüsselnden Textes entspricht, berechnet.

II.e
- **Erläutern** Sie ein Verfahren zum sicheren Austausch bzw. zur sicheren Generierung des Sitzungsschlüssels.
- **Untersuchen** Sie die Sicherheit des ChaCha20-Verfahrens. Gehen Sie dabei auch auf das Prinzip von Kerckhoffs´ ein.

(11 BE)

Anlage zur Aufgabe II: Virtual Private Network (Java-Version)

Anlage 1: XOR-Verknüpfung

```
1     xor   1     ->    0

1     xor   0     ->    1

0     xor   1     ->    1

0     xor   0     ->    0
```

Anlage 2: Evas Programm zur ChaCha20-Verschlüsselung

Die Datenstrukur `alphabet` ist eine `ArrayList`, wobei jedes Element der `ArrayList` vom Typ `Eintrag` ist (s. u.: ein Buchstabe und dazu eine `ArrayList` aus Binär-Ziffern). Im Folgenden ist die Datenstruktur als Pseudocode dargestellt:
`alphabet`:

```
[
    (A, [0, 1, 0, 0, 0, 0, 0, 1]),
    (B, [0, 1, 0, 0, 0, 0, 1, 0]),
    (C, [0, 1, 0, 0, 0, 0, 1, 1]),
    (D, [0, 1, 0, 0, 0, 1, 0, 0]),
    (E, [0, 1, 0, 0, 0, 1, 0, 1]),
    (F, [0, 1, 0, 0, 0, 1, 1, 0]),
    (G, [0, 1, 0, 0, 0, 1, 1, 1]),
    (H, [0, 1, 0, 0, 1, 0, 0, 0]),
    (I, [0, 1, 0, 0, 1, 0, 0, 1]),
    (J, [0, 1, 0, 0, 1, 0, 1, 0]),
    (K, [0, 1, 0, 0, 1, 0, 1, 1]),
    (L, [0, 1, 0, 0, 1, 1, 0, 0]),
    (M, [0, 1, 0, 0, 1, 1, 0, 1]),
    (N, [0, 1, 0, 0, 1, 1, 1, 0]),
    (O, [0, 1, 0, 0, 1, 1, 1, 1]),
    (P, [0, 1, 0, 1, 0, 0, 0, 0]),
    (Q, [0, 1, 0, 1, 0, 0, 0, 1]),
    (R, [0, 1, 0, 1, 0, 0, 1, 0]),
    (S, [0, 1, 0, 1, 0, 0, 1, 1]),
    (T, [0, 1, 0, 1, 0, 1, 0, 0]),
    (U, [0, 1, 0, 1, 0, 1, 0, 1]),
    (V, [0, 1, 0, 1, 0, 1, 1, 0]),
    (W, [0, 1, 0, 1, 0, 1, 1, 1]),
    (X, [0, 1, 0, 1, 1, 0, 0, 0]),
    (Y, [0, 1, 0, 1, 1, 0, 0, 1]),
    (Z, [0, 1, 0, 1, 1, 0, 1, 0])
]
```

Dazu sind folgende Funktionen gegeben (siehe folgende Seite):

```
1  public ArrayList<Integer> buchstabeInBinaer(String buchstabe,
2                                               ArrayList<Eintrag> datenListe) {
3          for (Eintrag eintrag : datenListe) {
4              if (eintrag.buchstabe.equals(buchstabe)) {
5                  return eintrag.binaerListe;
6              }
7          }
8          return new ArrayList<>();
9  }
10
11 public ArrayList<Integer> listeAlsBinaer(ArrayList<String> buchstabenListe,
12                                           ArrayList<Eintrag> datenListe) {
13         ArrayList<Integer> ausgabe = new ArrayList<>();
14         for (String buchstabe : buchstabenListe) {
15             ArrayList<Integer> binaerListe = buchstabeInBinaer(buchstabe,
16                                                                datenListe);
17             ausgabe.addAll(binaerListe);
18         }
19         return ausgabe;
20 }
21
22
23 public ArrayList<Integer> xorListe(
24                             ArrayList<Integer> schluesselListe,
25                             ArrayList<Integer> buchstabenAlsBinaerListe) {
26         ArrayList<Integer> ausgabe = new ArrayList<>();
27         for (int i = 0; i < schluesselListe.size(); i++) {
28             if (schluesselListe.get(i) == buchstabenAlsBinaerListe.get(i)) {
29                 ausgabe.add(0);
30             } else {
31                 ausgabe.add(1);
32             }
33         }
34         return ausgabe;
35 }
36
37 public ArrayList<Integer> verschluesselnXor(
38                                   ArrayList<Integer> streamingSchluessel,
39                                   ArrayList<String> klartextListe,
40                                   ArrayList<Eintrag> daten) {
41         ArrayList<Integer> klartextBinaer = listeAlsBinaer(klartextListe,
42                                                            daten);
43         return xorListe(streamingSchluessel, klartextBinaer);
44 }
```

Ein Eintrag wird durch folgende Klasse definiert:

```
1  public class Eintrag {
2      String buchstabe;
3      ArrayList<Integer> binaerListe;
4
5      public Eintrag(String buchstabe, ArrayList<Integer> binaere) {
6          this.buchstabe = buchstabe;
7          this.binaerListe = binaere;
8      }
9  }
```

Beispiel

`streamingSchluessel:`
`[0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1]`
`klartextListe: ['H', 'A', 'L']`
`alphabet: s. o.`

Beispielaufruf:

`verschluesselnXor(streamingSchluessel, klartextListe, alphabet)`

Ausgabe:

`[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]`

Aufgabe II: Virtual Private Network (Python-Version) 50 BE

Schwerpunkt: Datensicherheit in verteilten Systemen

Eva möchte von unterwegs auf ihre Daten zugreifen können, die sie zu Hause auf ihrem Server gespeichert hat. Dazu möchte sie eine Internetverbindung nutzen. Da sie auf ihrem Server auch vertrauliche Daten hat, möchte sie von ihrem Laptop aus, welchen sie auf ihren Reisen immer dabei hat, eine sogenannte VPN-Verbindung (VPN steht hier für „Virtual Private Network“; es handelt sich also in der Regel um eine nach außen hin verborgene und verschlüsselte Verbindung) zu ihrem Server zu Hause herstellen können. Folgenden Aufbau hat sie sich überlegt (Abbildung 1):

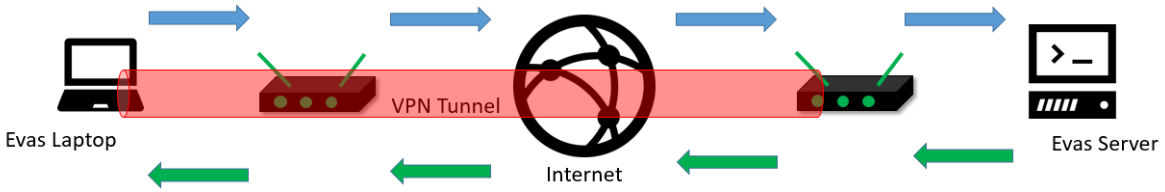


Abbildung 01

II.a **Stellen** Sie die Relevanz der drei folgenden Kriterien beim Einsatz einer verschlüsselten VPN-Verbindung **dar**:
- Authentizität
- Integrität
- Vertraulichkeit.

(10 BE)

Stellt Eva mit ihrem Laptop eine Verbindung zu ihrem Server zu Hause her, baut sie eine VPN-Verbindung im sogenannten Tunnelmodus zu ihrem Server auf. In vereinfachter Form sind die zu übertragenden Pakete inkl. der relevanten Header folgendermaßen aufgebaut (Abbildung 2):

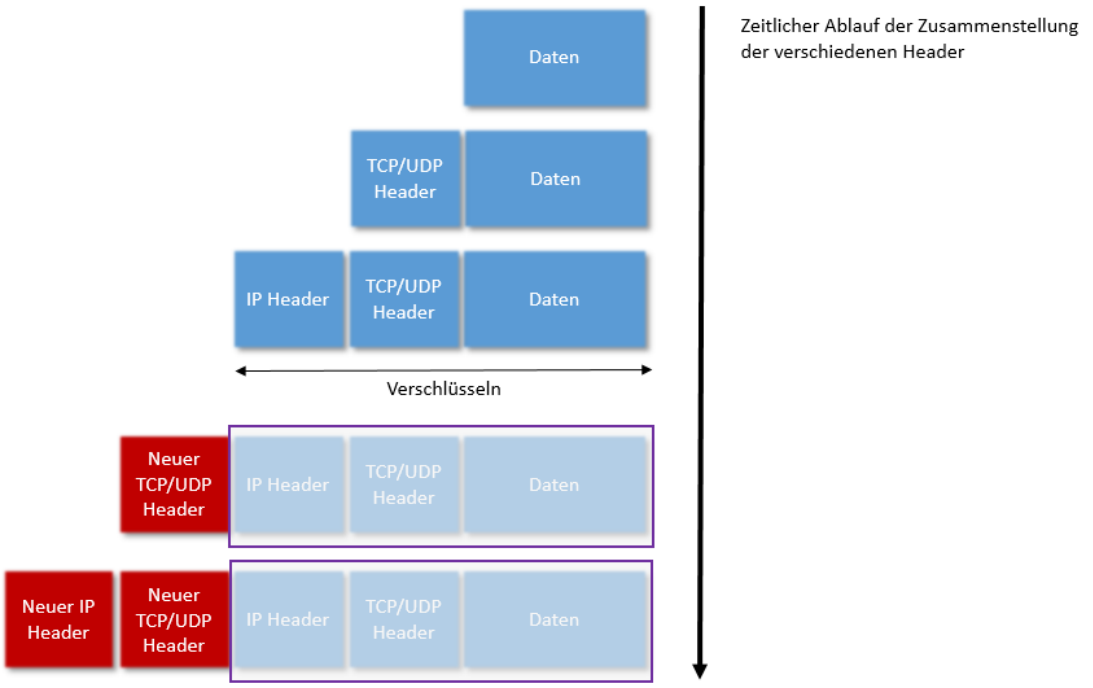


Abbildung 2

II.b
- **Beschreiben** Sie anhand der Abbildung 2, wie ein VPN-Tunnel entsteht und was man sich unter einem VPN-Tunnel vorzustellen hat.
- **Stellen** Sie **dar**, warum es zusätzlich zu dem Aufbau eines Tunnels nötig ist, die zu übermittelnden Daten zu verschlüsseln.

(11 BE)

Der VPN-Anbieter *Wireguard* nutzt den symmetrischen Verschlüsselungsalgorithmus ChaCha20. Er ist im Vergleich zu anderen modernen symmetrischen Verfahren (wie z. B. AES) sehr einfach, aber dennoch sehr sicher. Er funktioniert wie folgt:
Zunächst wird aus einem (256 Bit langen) symmetrischen Schlüssel ein sogenannter „Streaming-Schlüssel“ gebildet, der an die Länge des Klartextes angepasst wird. Anschließend wird der (binärcodierte) Klartext über die XOR-Verknüpfung mit dem „Streaming-Schlüssel“ verknüpft. (Für die XOR-Verknüpfung siehe Anlage 1).
Eva zweifelt das Verfahren an. Sie meint, dass die XOR-Verknüpfung nicht eindeutig wieder zu entschlüsseln ist.

II.c **Untersuchen** Sie, ob Evas Zweifel gerechtfertigt sind und die Entschlüsselung bei der XOR-Verknüpfung tatsächlich nicht eindeutig ist. Wie die XOR-Verknüpfung die Bits verarbeitet, lesen Sie in der Anlage 1.

(8 BE)

Eva hat begonnen, in einem kleinen Programm die ChaCha20-Verschlüsselung zu implementieren. Zunächst können nur Großbuchstaben verschlüsselt werden, z. B.:
Mit dem Klartext „HAL“
und dem folgenden Streaming-Schlüssel in passender Länge

```
[0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1]
```

würde sich die Chiffre

```
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]
```

ergeben.
Evas Programmcode finden Sie in der Anlage 2.

II.d **Erklären** Sie die Funktionsweise des Programms im Detail (zur Verarbeitung der Bits mit der XOR-Verknüpfung siehe Anlage 1).

(10 BE)

Der geheime, symmetrische Sitzungs-Schlüssel, den ChaCha20 verwendet, ist 256 Bit lang. Dieser Schlüssel wird für jede VPN-Sitzung neu berechnet und muss anschließend an beiden Tunnelenden bekannt bzw. vorhanden sein. Anschließend wird aus diesem Sitzungsschlüssel durch die Zugabe verschiedener Operationen ein pseudozufälliger Streaming-Schlüssel, der der Länge des zu verschlüsselnden Textes entspricht, berechnet.

II.e
- **Erläutern** Sie ein Verfahren zum sicheren Austausch bzw. zur sicheren Generierung des Sitzungsschlüssels.
- **Untersuchen** Sie die Sicherheit des ChaCha20-Verfahrens. Gehen Sie dabei auch auf das Prinzip von Kerckhoffs´ ein.

(11 BE)

Anlage zur Aufgabe II: Virtual Private Network (Python-Version)

Anlage 1: XOR-Verknüpfung

```
1     xor   1     ->    0

1     xor   0     ->    1

0     xor   1     ->    1

0     xor   0     ->    0
```

Anlage 2: Evas Programm zur ChaCha20-Verschlüsselung

Alphabet:

```
alphabet = [
    ('A', [0, 1, 0, 0, 0, 0, 0, 1]),
    ('B', [0, 1, 0, 0, 0, 0, 1, 0]),
    ('C', [0, 1, 0, 0, 0, 0, 1, 1]),
    ('D', [0, 1, 0, 0, 0, 1, 0, 0]),
    ('E', [0, 1, 0, 0, 0, 1, 0, 1]),
    ('F', [0, 1, 0, 0, 0, 1, 1, 0]),
    ('G', [0, 1, 0, 0, 0, 1, 1, 1]),
    ('H', [0, 1, 0, 0, 1, 0, 0, 0]),
    ('I', [0, 1, 0, 0, 1, 0, 0, 1]),
    ('J', [0, 1, 0, 0, 1, 0, 1, 0]),
    ('K', [0, 1, 0, 0, 1, 0, 1, 1]),
    ('L', [0, 1, 0, 0, 1, 1, 0, 0]),
    ('M', [0, 1, 0, 0, 1, 1, 0, 1]),
    ('N', [0, 1, 0, 0, 1, 1, 1, 0]),
    ('O', [0, 1, 0, 0, 1, 1, 1, 1]),
    ('P', [0, 1, 0, 1, 0, 0, 0, 0]),
    ('Q', [0, 1, 0, 1, 0, 0, 0, 1]),
    ('R', [0, 1, 0, 1, 0, 0, 1, 0]),
    ('S', [0, 1, 0, 1, 0, 0, 1, 1]),
    ('T', [0, 1, 0, 1, 0, 1, 0, 0]),
    ('U', [0, 1, 0, 1, 0, 1, 0, 1]),
    ('V', [0, 1, 0, 1, 0, 1, 1, 0]),
    ('W', [0, 1, 0, 1, 0, 1, 1, 1]),
    ('X', [0, 1, 0, 1, 1, 0, 0, 0]),
    ('Y', [0, 1, 0, 1, 1, 0, 0, 1]),
    ('Z', [0, 1, 0, 1, 1, 0, 1, 0]),
]
```

Dazu sind folgende Funktionen gegeben (siehe folgende Seite):

```
1   def buchstabeInBinaer(buchstabe, alphabet):
2       for eintrag in alphabet:
3           if buchstabe == eintrag[0]:
4               return eintrag[1]
5       return ['Fehler']
6
7   def listeAlsBinaer(buchstabenListe, alphabet):
8       ausgabe = []
9       for buchstabe in buchstabenListe:
10          ausgabe.append(buchstabeInBinaer(buchstabe, alphabet))
11      return ausgabe
12
13  def xorListe(schluesselListe, buchstabenListe):
14      ausgabe = []
15      for i in range(len(schluesselListe)):
16          if schluesselListe[i] == buchstabenListe[i]:
17              ausgabe.append(0)
18          else:
19              ausgabe.append(1)
20      return ausgabe
21
22  def verschluesselnXor(streamingSchluessel, klartextListe, alphabet):
23      klartextBinaer = listeAlsBinaer(klartextListe, alphabet)
24      ausgabe = xorListe (streamingSchluessel, klartextBinaer)
25      return ausgabe
```

Beispielaufruf:

```
streamingSchluessel = [0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 1, 1]

klartextListe = ['H', 'A', 'L']

print(verschluesselnXor(streamingSchluessel, klartextListe, alphabet))
```

Ausgabe:

```
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]
```

Aufgabe III: Dame (Java-Version) 50 BE

Schwerpunkt: Intelligente Suchverfahren

Das Dame-Spiel ist ein Zweipersonenspiel an einem Spielbrett. Das Spielbrett entspricht einem normalen Schachbrett (siehe Abbildung 1 und für eine ausführliche Beschreibung des Spiels die Anlage 1).
Alle schwarzen und weißen Steine werden allein auf den schwarzen Feldern (diagonal) und immer in Richtung der gegnerischen Seite bewegt, um dort die Grundlinie zu erreichen.
Wir betrachten das Spiel nur aus der Sicht des Spielers der weißen Steine. Er kann bei einer Position wie in Abbildung 2 den schwarzen Stein durch Überspringen „schlagen“, um ihn vom Feld zu entfernen.

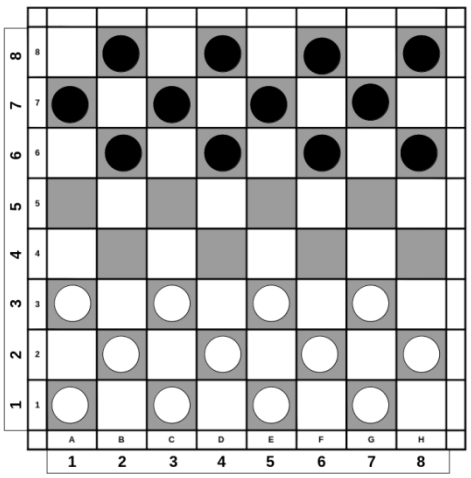


Abbildung 1

Springen und Schlagen:

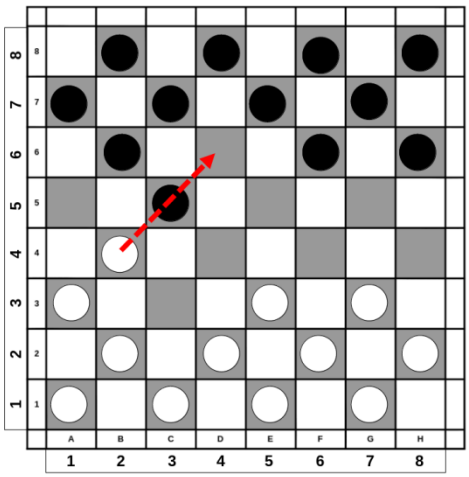


Abbildung 2

Zur Beschreibung der Positionen werden statt wie beim Schach üblich der Buchstaben in dieser Aufgabe zur Vereinfachung die Zahlen von 1 bis 8 verwendet.
Als Datenstruktur zum Verwalten des Zustands des Spielfeldes verwenden wir zwei Listen (`ArrayList`):

```
private ArrayList<int[]> weisse, schwarze;
```

Diesen Listen werden für den Startzustand (siehe hierzu auch Abbildung 1) jeweils nacheinander die Koordinaten der von den weißen beziehungsweise schwarzen Steinen belegten Felder hinzugefügt. Die Felder werden als zweidimensionale Integer-Arrays gespeichert, also

`{1,1}, {3,1}, ... , {7,3}` für die weißen Steine und

`{2,6}, {4,6}, ... , {8,8}` für die schwarzen Steine.

III.a
- **Erläutern** Sie die obige Datenstruktur `ArrayList` (siehe Seite 1).
- Mit der Definition

```
private int[] linksOben = {-1,1};
```

werden die Daten für das Verschieben eines Steins nach links oben angegeben.
Geben Sie die fehlenden Definitionen für die anderen drei möglichen Bewegungen beim Setzen der Steine **an**.
- **Erläutern** Sie die Funktionsweise der im Folgenden angegebenen Hilfsfunktion `istSchwarzesFeld`:

```
1  /* Hilfsfunktion  istSchwarzesFeld */
2      public boolean istSchwarzesFeld(int[] feld) {
3          boolean imFeld = (feld[0]>0 && feld[1]>0
4                            && feld[0]<=8 && feld[1]<=8);
5          return imFeld && istGeradeZahl(feld[0]+feld[1]);
6      }
7
8      private boolean istGeradeZahl(int zahl) {
9          return zahl%2==0;
10     }
```

(12 BE)

Erreicht ein Spielstein die gegnerische Grundlinie, wird er zu einer „Dame“ aufgewertet. Diese Dame ist sehr viel wertvoller, da sie sich auch diagonal rückwärts bewegen kann und bei ihrem Ziehen auch mehrere freie Felder in einem Zug überwinden kann.
Ein vorrangiges Ziel eines Spielers ist daher, nach einem Weg zu suchen, auf dem einer seiner Steine die gegnerische Grundlinie erreicht und zu einer Dame wird. Verwenden Sie für die folgende Teilaufgabe den im folgenden Bild dargestellten Zustand des Spielfelds.

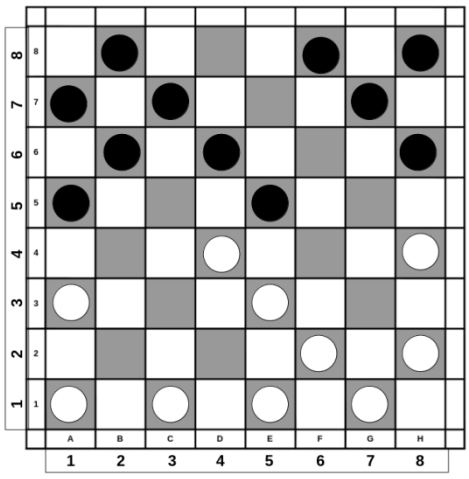


Abbildung 3

III.b
- **Beschreiben** Sie am Beispiel des weißen Steins auf dem Feld D 4 (also dem Feld mit der Kennzeichnung `[4, 4]`) ein Suchverfahren, mit dem der weiße Spieler beim dargestellten Zustand in der obigen Abbildung 3 in mehreren Schritten einen Weg zur gegnerischen Grundlinie finden kann.
Hinweis: Gehen Sie hier vereinfachend davon aus, dass der Gegner inzwischen nicht selbst mit seinen Steinen ziehen kann.
- **Erläutern** Sie, dass das Problem komplizierter wird, wenn der Gegner (wie im Spiel üblich) jeweils selbst auch mit seinen Steinen ziehen kann.

(9 BE)

Die in der Anlage 2 angegebene Funktion untersucht, ob der übergebene Stein springen kann, und gibt in dem Fall das Zielfeld zurück.

III.c **Analysieren** Sie die in Anlage 2 angegebene Funktion `weisserKannSpringen`.

(8 BE)

In den folgenden Teilaufgaben wird nicht mehr die Dame des gleichnamigen Dame-Spiels, sondern die Dame beim Schachspiel thematisiert. Beim Schachspiel führen die Spielmöglichkeiten der Dame auf eine andere Problemstellung. Hier kann die Dame nicht nur diagonal, sondern auch waagerecht (also von links nach rechts oder rechts nach links) oder senkrecht (also von vorn nach hinten bzw. von hinten nach vorn, im Bild von oben nach unten bzw. von unten nach oben) beliebig weit bewegt werden, um eine andere Spielfigur zu schlagen, wenn sie diese trifft. Wie beim Schach üblich, darf sie zudem alle Felder besetzen, also sowohl weiße als auch schwarze. Anders als beim Damespiel darf sie die gegnerischen Spielfiguren zum Schlagen aber nicht überspringen, stattdessen nimmt sie deren Platz ein.

In Loslösung vom eigentlichen Schachspiel mit seinen Regeln stellt sich hier nun die Frage, ob sich mehrere solcher Damen (unabhängig von ihrer Farbe) so auf dem Spielfeld positionieren lassen, dass sie sich nicht untereinander bedrohen, also nicht in derselben Zeile, Spalte oder Diagonale stehen (siehe Abbildung 4).

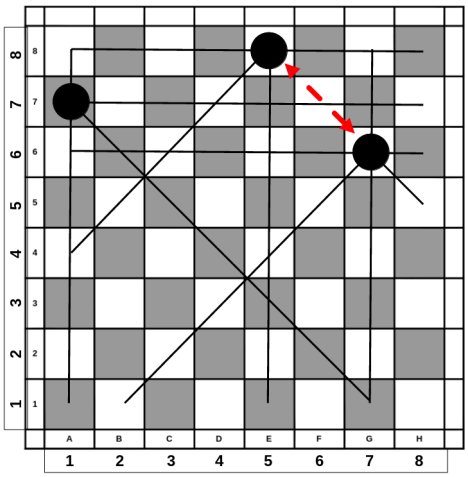


Abbildung 4

III.d
- **Geben** Sie für ein Spielfeld mit 4-mal-4 Feldern (also vier Zeilen und vier Spalten) eine Belegung **an**, bei der sich vier Damen nicht bedrohen (sich also nicht gegenseitig schlagen können), und **erläutern** Sie diese.
- **Erläutern** Sie das prinzipielle Vorgehen, um mit Hilfe der Tiefensuche eine Belegung mit acht Damen auf einem 8-mal-8-Feld zu finden.

(13 BE)

III.e **Analysieren** Sie die Funktionsweise der folgenden Funktion `freiesFeld`, die für jede der vorhandenen Damen ein Feld sucht, auf dem sie nicht bedroht wird (also nicht geschlagen werden kann).

```
1   /**
2    * freiesFeld
3    */
4   public int[] freiesFeld(){
5       for (int[] feld : felder)
6           if (ok(feld)) return feld;
7       return null;
8   }
9
10  /**
11   * Hilfsfunktion ok
12   */
13  public boolean ok(int[] feld){
14      for (int[] dame : damen)
15          if (bedroht(dame, feld))  return false;
16      return true;
17  }
18
19  /**
20   * Hilfsfunktionen für Zeile und Spalte
21   */
22  public int zeile(int[] feld) { return feld[1]; }
23  public int spalte(int[] feld) { return feld[0]; }
24
25  /**
26   * Hilfsfunktion bedroht
27   */
28  public boolean bedroht(int[] dame, int[] feld) {
29      if (zeile(dame) == zeile(feld)) return true;
30      if (spalte(dame) == spalte(feld)) return true;
31      if (spalte(dame)-zeile(dame) == (spalte(feld)-zeile(feld)))
32          return true;
33      if (spalte(dame)+zeile(dame) == (spalte(feld)+zeile(feld)))
34          return true;
35      return false;
36  }
```

(8 BE)

Anlagen zur Aufgabe III Dame (Java-Version)

Anlage 1: *Ausführliche Spielbeschreibung*

Beim Dame-Spiel spielen zwei Spielerinnen oder Spieler gegeneinander, die sich am Spielbrett, welches einem normalen Schachbrett entspricht (siehe Abbildung 1), gegenübersitzen.

Es gibt einfache flache Spielsteine, für die Spielerin oder den Spieler die weißen, für die Gegenerin oder den Gegner die schwarzen.

Alle Steine werden allein auf den schwarzen Feldern bewegt, so dass sie sich nur diagonal bewegen dürfen. Die einfachen Spielsteine werden dabei jeweils ein Feld allein in der Richtung zum Gegner beziehungsweise der Gegnerin hin bewegt.

Wenn ein eigener Stein direkt vor einem gegnerischen Spielstein steht und das Feld dahinter frei ist, kann der gegnerische Spielstein „geschlagen“ werden, indem man ihn überspringt. Der geschlagene Stein wird dann vom Spielfeld entfernt (siehe Abbildung 2).

Dabei kann es vorkommen, dass nach dem ersten Sprung weitergesprungen werden kann, so dass mehrere gegnerische Steine in einem Zug geschlagen werden können.

Wird die Möglichkeit zum Springen nicht wahrgenommen, kann der Gegner beziehungsweise die Gegnerin den entsprechenden Stein vom Spielfeld entfernen.

Das Spiel ist gewonnen, wenn die Gegnerin bzw. der Gegner nicht mehr ziehen oder springen kann.

Die Felder werden üblicherweise mit Buchstaben für die Spalten und Zahlen für die Zeilen benannt, also beispielsweise A1 oder B3 bis H8 (wahlweise mit kleinen Buchstaben). Zur Vereinfachung werden für die Aufgaben aber keine Buchstaben (Symbole) zur Benennung der Felder angewendet, sondern nur die Spaltenindexwerte und die Zeilenindexwerte, jeweils von 1 bis 8.

Anlage 2: *Funktion für Teilaufgabe III.c*

```
1   private ArrayList<int[]> weisse, schwarze, richtungen;
2
3   public int[] weisserKannSpringen(int[] stein) {
4       if (!istWeisser(stein)) return null;
5       int[] ziel = sprungziel(stein, richtungen.get(0));
6       if (istSchwarzer(neuesFeld(stein, richtungen.get(0))))
7           if (feldFrei(ziel))
8               return ziel;
9       ziel = sprungziel(stein, richtungen.get(1));
10      if (istSchwarzer(neuesFeld(stein, richtungen.get(1))))
11          if (feldFrei(ziel))
12              return ziel;
13      return null;
14  }
15
16  /**
17   * Hilfsfunktion sprungziel
18  */
19  private int[] sprungziel(int[] stein, int[] richtung) {
20      int[] ueber=neuesFeld(stein, richtung);
21      int[] auf=neuesFeld(ueber, richtung);
22      return auf;
23  }
24
25  /**
26   * Hilfsfunktion istWeisser
27   * prüft, ob das Feld von einem weißen Stein besetzt ist
28  */
29  public boolean istWeisser(int[] derStein) {
30      ...
31  }
32
33  /**
34   * Hilfsfunktion istSchwarzer
35   * prüft, ob das Feld von einem schwarzen Stein besetzt ist
36  */
37  public boolean istSchwarzer(int[] derStein) {
38      ...
39  }
40
41
42  /**
43   * Hilfsfunktion neuesFeld
44   * gibt das neue Feld in der übergebenen Richtung zurück
45  */
46  public int[] neuesFeld(int[] stein, int[] richtung) {
47      ...
48  }
49
50  /**
51   * Hilfsfunktion feldFrei
52   * prueft, ob das uebergebene Feld beim aktuellen Zustand frei ist
53  */
54      public boolean feldFrei(int[] feld) {
55          ...
56      }
```

Aufgabe III: Dame (Python-Version) 50 BE

Schwerpunkt: Intelligente Suchverfahren

Das Dame-Spiel ist ein Zweipersonenspiel an einem Spielbrett. Das Spielbrett entspricht einem normalen Schachbrett (siehe Abbildung 1 und für eine ausführliche Beschreibung des Spiels die Anlage 1).
Alle schwarzen und weißen Steine werden allein auf den schwarzen Feldern (diagonal) und immer in Richtung der gegnerischen Seite bewegt, um dort die Grundlinie zu erreichen.
Wir betrachten das Spiel nur aus der Sicht des Spielers der weißen Steine. Er kann bei einer Position wie in Abbildung 2 den schwarzen Stein durch Überspringen „schlagen“, um ihn vom Feld zu entfernen.

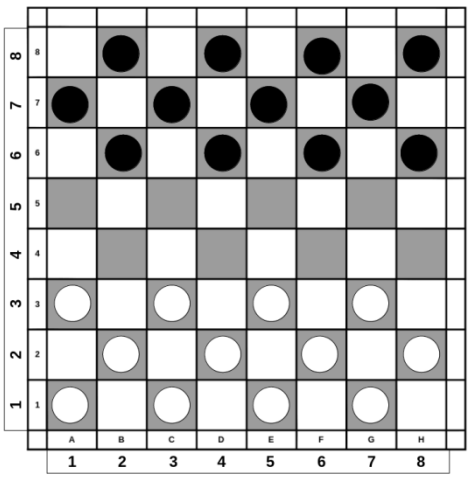


Abbildung 1

Springen und Schlagen:

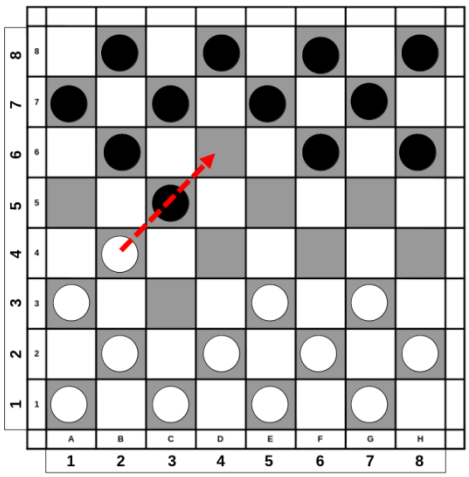


Abbildung 2

Zur Beschreibung der Positionen werden statt wie beim Schach üblich der Buchstaben in dieser Aufgabe zur Vereinfachung die Zahlen von 1 bis 8 verwendet.
Die Daten des Spielzustands werden in einer Variablen `zustand` gehalten. Als Datenstruktur zum Verwalten dieses Zustands des Spielfelds wird die folgende Liste verwendet, die so wie angegeben den Startzustand beschreibt (siehe auch Abbildung 1):

```
zustand =
[ [(1,1),(3,1),(5,1),(7,1),(2,2),(4,2),(6,2),(8,2),(1,3),(3,3),(5,3),(7,3)],
  [(2,6),(4,6),(6,6),(8,6),(1,7),(3,7),(5,7),(7,7),(2,8),(4,8),(6,8),(8,8)] ]
```

III.a
- **Erläutern** Sie die obige Datenstruktur.
- Mit der Definition

  ```
  links_oben = (-1, 1)
  ```

  werden die Daten für das Verschieben eines Steins nach links oben angegeben.
  Geben Sie die fehlenden Definitionen für die anderen drei möglichen Bewegungen beim Setzen der Steine **an**.
- **Erläutern** Sie die Funktionsweise der im Folgenden angegebenen Hilfsfunktion `ist_schwarzes_feld`:

```
1  ### Hilfsfunktionen ist_schwarzes_feld und ist_gerade_zahl
2  def ist_schwarzes_feld(feld):
3      im_feld = feld[0]>0 and feld[1]>0 and feld[0]<=8 and feld[1]<=8
4      return im_feld and ist_gerade_zahl(feld[0]+feld[1])
5  def ist_gerade_zahl(zahl):
6      return zahl%2 == 0
```

(12 BE)

Erreicht ein Spielstein die gegnerische Grundlinie, wird er zu einer „Dame“ aufgewertet. Diese Dame ist sehr viel wertvoller, da sie sich auch diagonal rückwärts bewegen kann und bei ihrem Ziehen auch mehrere freie Felder in einem Zug überwinden kann.
Ein vorrangiges Ziel eines Spielers ist daher, nach einem Weg zu suchen, auf dem einer seiner Steine die gegnerische Grundlinie erreicht und zu einer Dame wird. Verwenden Sie für die folgende Teilaufgabe den im folgenden Bild dargestellten Zustand des Spielfelds.

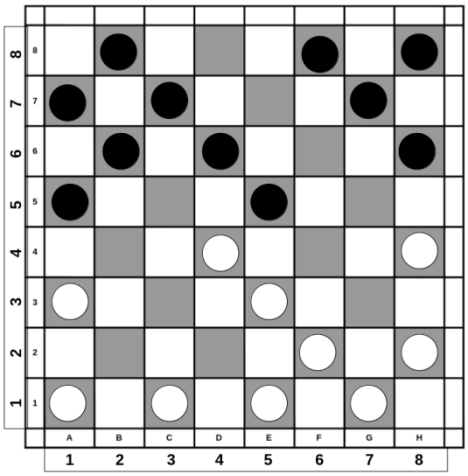


Abbildung 3

III.b
- **Beschreiben** Sie am Beispiel des weißen Steins auf dem Feld D 4 (also dem Feld mit der Kennzeichnung `[4, 4]`) ein Suchverfahren, mit dem der weiße Spieler beim dargestellten Zustand in der obigen Abbildung 3 in mehreren Schritten einen Weg zur gegnerischen Grundlinie finden kann.
  Hinweis: Gehen Sie hier vereinfachend davon aus, dass der Gegner inzwischen nicht selbst mit seinen Steinen ziehen kann.
- **Erläutern** Sie, dass das Problem komplizierter wird, wenn der Gegner (wie im Spiel üblich) jeweils selbst auch mit seinen Steinen ziehen kann.

(9 BE)

Die in der Anlage 2 angegebene Funktion untersucht, ob der übergebene Stein springen kann, und gibt in dem Fall das Zielfeld zurück.

III.c **Analysieren** Sie die in Anlage 2 angegebene Funktion `weisserKannSpringen`.

(8 BE)

In den folgenden Teilaufgaben wird nicht mehr die Dame des gleichnamigen Dame-Spiels, sondern die Dame beim Schachspiel thematisiert. Beim Schachspiel führen die Spielmöglichkeiten der Dame auf eine andere Problemstellung. Hier kann die Dame nicht nur diagonal, sondern auch waagerecht (also von links nach rechts oder rechts nach links) oder senkrecht (also von vorn nach hinten bzw. von hinten nach vorn, im Bild von oben nach unten bzw. von unten nach oben) beliebig weit bewegt werden, um eine andere Spielfigur zu schlagen, wenn sie diese trifft. Wie beim Schach üblich, darf sie zudem alle Felder besetzen, also sowohl weiße als auch schwarze. Anders als beim Damespiel darf sie die gegnerischen Spielfiguren zum Schlagen aber nicht überspringen, stattdessen nimmt sie deren Platz ein.

In Loslösung vom eigentlichen Schachspiel mit seinen Regeln stellt sich hier nun die Frage, ob sich mehrere solcher Damen (unabhängig von ihrer Farbe) so auf dem Spielfeld positionieren lassen, dass sie sich nicht untereinander bedrohen, also nicht in der selben Zeile, Spalte oder Diagonale stehen (siehe Abbildung 4).

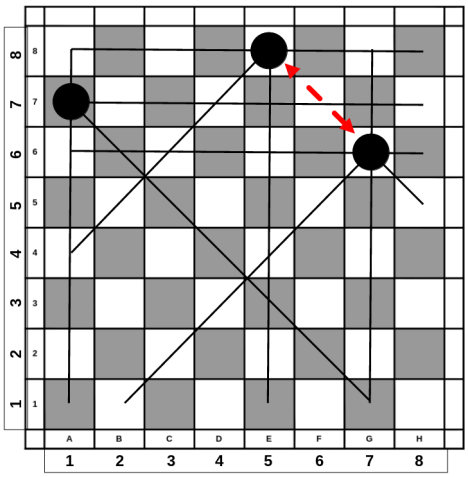


Abbildung 4

III.d
- **Geben** Sie für ein Spielfeld mit 4-mal-4 Feldern (also vier Zeilen und vier Spalten) eine Belegung **an**, bei der sich vier Damen nicht bedrohen (sich also nicht gegenseitig schlagen können), und **erläutern** Sie diese.
- **Erläutern** Sie das prinzipielle Vorgehen, um mit Hilfe der Tiefensuche eine Belegung mit acht Damen auf einem 8-mal-8-Feld zu finden.

(13 BE)

III.e **Analysieren** Sie die Funktionsweise der folgenden Funktion `freies_feld`, die für jede der vorhandenen Damen ein Feld sucht, auf dem sie nicht bedroht wird (also nicht geschlagen werden kann).

```
1  def freies_feld(alle_damen, alle_felder):
2      for feld in alle_felder:
3          if ok(alle_damen, feld):
4              return feld
5      return []
6
7  ### Hilfsfunktionen:
8  def ok(damen, feld):
9      for dame in damen:
10         if bedroht(dame, feld):
11             return False
12     return True
13
14 def zeile(feld): return feld[1]
15 def spalte(feld): return feld[0]
16
17 ### Hilfsfunktion bedroht
18 ### prueft, ob die dame das feld bedroht
19 def bedroht(dame, feld):
20     if zeile(dame) == zeile(feld): return True
21     if spalte(dame) == spalte(feld): return True
22     if (spalte(dame)-zeile(dame)) == (spalte(feld)-zeile(feld)): return True
23     if (spalte(dame)+zeile(dame)) == (spalte(feld)+zeile(feld)): return True
24     return False
```

(8 BE)

Anlagen zur Aufgabe III Dame (Python-Version)

Anlage 1: *Ausführliche Spielbeschreibung*

Beim Dame-Spiel spielen zwei Spielerinnen oder Spieler gegeneinander, die sich am Spielbrett, welches einem normalen Schachbrett entspricht (siehe Abbildung 1), gegenübersitzen.

Es gibt einfache flache Spielsteine, für die Spielerin oder den Spieler die weißen, für die Gegenerin oder den Gegner die schwarzen.

Alle Steine werden allein auf den schwarzen Feldern bewegt, so dass sie sich nur diagonal bewegen dürfen. Die einfachen Spielsteine werden dabei jeweils ein Feld allein in der Richtung zum Gegner beziehungsweise der Gegnerin hin bewegt.

Wenn ein eigener Stein direkt vor einem gegnerischen Spielstein steht und das Feld dahinter frei ist, kann der gegnerische Spielstein „geschlagen“ werden, indem man ihn überspringt. Der geschlagene Stein wird dann vom Spielfeld entfernt (siehe Abbildung 2).

Dabei kann es vorkommen, dass nach dem ersten Sprung weitergesprungen werden kann, so dass mehrere gegnerische Steine in einem Zug geschlagen werden können.

Wird die Möglichkeit zum Springen nicht wahrgenommen, kann der Gegner beziehungsweise die Gegnerin den entsprechenden Stein vom Spielfeld entfernen.

Das Spiel ist gewonnen, wenn die Gegnerin bzw. der Gegner nicht mehr ziehen oder springen kann.

Die Felder werden üblicherweise mit Buchstaben für die Spalten und Zahlen für die Zeilen benannt, also beispielsweise A1 oder B3 bis H8 (wahlweise mit kleinen Buchstaben). Zur Vereinfachung werden für die Aufgaben aber keine Buchstaben (Symbole) zur Benennung der Felder angewendet, sondern nur die Spaltenindexwerte und die Zeilenindexwerte, jeweils von 1 bis 8.

Anlage 2: *Funktion für Teilaufgabe III.c*

```
### ----- weisser_kann_springen -----
def weisser_kann_springen(stein, richtungen, zustand):
    ### richtungen: (rechts_oben,links_oben)
    if not ist_weisser(stein, zustand):
        return None
    ziel=sprungziel(stein, richtungen[0])
    if ist_schwarzer(neues_feld(stein, richtungen[0]), zustand):
        if feld_frei(ziel, zustand):
            return ziel
    ziel=sprungziel(stein, richtungen[1])
    if ist_schwarzer(neues_feld(stein, richtungen[1]), zustand):
        if feld_frei(ziel, zustand):
            return ziel
    return None

### Definition der Richtungen; abhängig von der verwendeten Datenstruktur
### die ersten beiden für weiss
### die anderen für schwarz
richtungen = (rechts_oben,links_oben,rechts_unten,links_unten)

### Hilfsfunktion ist_weisser
### prüft, ob das Feld von einem weißen Stein besetzt ist
def ist_weisser(stein, zustand):
    return stein in zustand[0]

### Hilfsfunktion  ist_schwarzer
### prüft, ob das Feld von einem schwarzen Stein besetzt ist
def ist_schwarzer(stein, zustand):
    return stein in zustand[1]

### Hilfsfunktion neues_Feld
### gibt das neue Feld in der übergebenen Richtung zurück
def neues_feld(stein, richtung):
    return stein[0]+richtung[0],stein[1]+richtung[0]
### Hilfsfunktion sprungziel
def sprungziel(stein, richtung):
    ueber=neues_feld(stein, richtung)
    auf=neues_feld(ueber, richtung)
    return auf

### Hilfsfunktion feld_frei
### prueft, ob das uebergebene Feld beim aktuellen Zustand frei ist
def feld_frei(feld, zustand):
    if not ist_schwarzes_feld(feld): return False
    elif ist_weisser(feld, zustand): return False
    elif ist_schwarzer(feld, zustand): return False
    return True
```