Prof. Dr. J. Giesl
J.-C. Kassing, M. Leven Rosarius, N. Lommen, E. Meyer

Allgemeine Hinweise:

- Die **Deadline** zur **Abgabe** der Hausaufgaben ist am **Donnerstag, den 20.11.2025, um 14 Uhr**.
- Der **Workflow** sieht wie folgt aus. Die Abgabe der Hausaufgaben erfolgt **im Moodle-Lernraum** und kann nur in **Zweiergruppen** stattfinden. Dabei müssen die Abgabepartner*innen **dasselbe Tutorium** besuchen. Nutzen Sie ggf. das entsprechende **Forum** im Moodle-Lernraum, um eine*n Abgabepartner*in zu finden. Es darf **nur ein*e** Abgabepartner*in die Abgabe hochladen. Diese*r muss sowohl die **Lösung** als auch den **Quellcode** der Programmieraufgaben hochladen. Die Bepunktung wird dann von uns für **beide** Abgabepartner*innen **separat** im Lernraum eingetragen. Die Feedbackdatei ist jedoch nur dort sichtbar, wo die Abgabe hochgeladen wurde und muss innerhalb des Abgabepaars **weitergeleitet** werden.
- Die **Lösung** muss als PDF-Datei hochgeladen werden. Damit die Punkte beiden Abgabepartner*innen zugeordnet werden können, müssen **oben** auf der **ersten Seite** Ihrer Lösung die **Namen**, die **Matrikelnummern** sowie die **Nummer des Tutoriums** von **beiden** Abgabepartner*innen angegeben sein.
- Der **Quellcode** der Programmieraufgaben muss als `.zip`-Datei hochgeladen werden und **zusätzlich** in der PDF-Datei mit Ihrer Lösung enthalten sein, sodass unsere Hiwis ihn mit Feedback versehen können. Auf diesem Blatt muss Ihre Codeabgabe Ihren vollständigen Java-Code in Form von `.java`-Dateien enthalten. Aus dem Lernraum heruntergeladene Klassen dürfen nicht mit abgegeben werden.

  Stellen Sie sicher, dass Ihr Programm von `javac` **in der Version 25 akzeptiert** wird. Generell sollten alle Programme für alle Eingaben terminieren, solange in der Spezifikation (bzw. der Aufgabenstellung) nicht explizit etwas anderes verlangt wird!
- Einige Hausaufgaben müssen im Spiel Codescape gelöst werden. Klicken Sie dazu im Lernraum rechts im Block "Codescape" auf den angegebenen Link. Diese Aufgaben werden getrennt von den anderen Hausaufgaben gewertet.

Übungsaufgabe 1 (Überblickswissen):

a) In der Vorlesung wurde das Java-Feature `record`-Klassen eingeführt. Was sind die Vorteile und was sind die Nachteile von diesen Klassen gegenüber gewöhnlichen Klassen?

b) Warum ist es gerade bei Attributen sinnvoll, diese, soweit möglich, mit dem `final`-Schlüsselwort zu deklarieren?

c) Was ist der Unterschied zwischen `int` und `Integer` in Java?

d) Macht es bei Attributen, für die eine Klasse einen Getter zur Verfügung stellt, einen Unterschied, mit welchem Zugriffsmodifikator diese versehen sind?

e) Wie werden die Attribute eines Objekts beim Aufruf des automatisch erzeugten, parameterlosen Konstruktors gesetzt?

Übungsaufgabe 2 (Programmierung mit Datenabstraktion):

In dieser Aufgabe wird eine Klasse `Point` und eine weitere Klasse `LineSegment` implementiert. Die erste Klasse repräsentiert hierbei einen (2-dimensionalen) Punkt. Die zweite Klasse stellt die Strecke zwischen zwei Punkten dar. Eine Strecke ist hier also eine *endliche* Gerade auf der 2-dimensionalen Ebene.
Beachten Sie in allen Teilaufgaben die *Prinzipien der Datenkapselung*. Überlegen Sie sich außerdem, welche Methoden statisch sein müssen. Verwenden Sie die exakt gleichen Methoden- und Klassennamen wie vorgegeben.

Hinweise:

- Beachten Sie, dass durch Rundungen in dieser Aufgabe die Ergebnisse von manchen Berechnungen von den tatsächlichen Ergebnissen leicht abweichen können.

a) Schreiben Sie die Record-Klasse `Point`. Diese Klasse hat hierbei die beiden Attribute `int x` und `int y`.

b) Schreiben Sie die Methode `String toString()`, welche das aktuelle Objekt der Klasse `Point` mit Koordinaten `x` und `y` als String "`(x,y)`" zurückgibt. Implementieren Sie zusätzlich eine Methode `double norm2()`, welche die 2-Norm des aktuellen Punktes berechnet. D.h. für den Punkt `(x,y)` soll $\sqrt{x^2 + y^2}$ berechnet werden.

Hinweise:

- Sie können die Methode `double Math.sqrt(double a)` verwenden, um die Quadratwurzel zu berechnen.

c) Schreiben Sie die Klasse `LineSegment`, welche eine Strecke repräsentiert. Hierzu sollen in der Klasse zwei Punkte gespeichert werden, welche die Endpunkte der Strecke bilden. Implementieren Sie die folgenden Konstruktoren:

i) Ein Konstruktor soll die beiden Endpunkte der neuen Strecke übergeben bekommen.

ii) Ein Konstruktor soll in dieser Reihenfolge vier `int`-Werte `x1`, `y1`, `x2` und `y2` übergeben bekommen. Hiermit sollen entsprechend die beiden Endpunkte `(x1,y1)` und `(x2,y2)` der neuen Strecke erzeugt werden.

iii) Schreiben Sie einen Konstruktor, welcher den `int`-Parameter `max` übergeben bekommt. Erzeugen Sie eine Strecke, indem Sie zufällig die Koordinaten der Endpunkte aus dem Intervall 0 bis `max` -1 wählen.

Hinweise:

- Zufallszahlen von 0 bis `max` -1 können Sie nach der Initialisierung `Random rn = new Random();` mit `rn.nextInt(max);` generieren. Um die Klasse `Random` verwenden zu können, müssen Sie diese mittels `import java.util.Random;` am Anfang in der Datei der Klasse `LineSegment` importieren.

iv) Definieren Sie den Konstruktor `LineSegment(int max, int length)`. Hierbei sollen die Koordinaten eines Endpunktes im Intervall 0 bis `max` -1 zufällig gewählt werden. Der zweite Endpunkt soll nun auch zufällig im Intervall 0 bis `max` -1 gewählt sein, jedoch mit Abstand `length` zum ersten Endpunkt.

Hinweise:

- Möchten Sie einen zufälligen Punkt mit Abstand `length` zu dem Punkt `(x,y)` berechnen, dann wählen Sie einen zufälligen Winkel `angle` aus dem Intervall $[0, 2\pi)$ (z.B.: mit `rn.nextInt(360) * Math.PI / 180`). Dann hat der Punkt `(u,v)` mit `u = x + length * Math.cos(angle)` und `v = y + length * Math.sin(angle)` gerade Abstand `length` zu `(x,y)`.
- Da wir nur `int`-Koordinaten betrachten, sollte der Punkt `(u,v)` dann in `((int) u, (int) v)` konvertiert werden, auch wenn dadurch der Abstand geringfügig verändert wird.
- Obiges Verfahren liefert nicht zwingend einen Punkt im Intervall 0 bis `max`-1. Wiederholen Sie das Verfahren solange, bis Sie einen Punkt erhalten, welcher im Intervall liegt.

d) Schreiben Sie in der Klasse `LineSegment` Selektor-Methoden `setA` und `setB` für beide Endpunkte der Strecke. Implementieren Sie ebenfalls die Getter-Methoden `getA` und `getB` für beide Punkte. Implementieren Sie außerdem die Methode `String toString()`, sodass die Strecke zwischen den Punkten `(x1,y1)` und `(x2,y2)` als `String` "`(x1,y1) -- (x2,y2)`" zurückgegeben wird.

e) Schreiben Sie in der Klasse `LineSegment` die Methode `LineSegment[] spawnParallel(int distance, int n)`. Diese erzeugt `n` horizontale, parallele Strecken, beginnend bei der Strecke `(0,0) -- (distance,0)` und endend mit der Strecke `(0,distance) -- (distance,distance)`, mit Abstand $\lfloor \frac{\texttt{distance}}{\texttt{n}-1} \rfloor$ für $\texttt{n} > 1$. Für jede natürliche Zahl x bezeichnet $\lfloor x \rfloor$ die größte natürliche Zahl mit $\lfloor x \rfloor \leq x$. Für $\texttt{n} \leq 1$ kann sich Ihre Methode beliebig verhalten.

f) Schreiben Sie in der Klasse `LineSegment` die Methode `boolean intersectHorizontal(LineSegment l)`, welche genau dann `true` zurückgibt, wenn `this` einen Schnittpunkt mit der horizontalen *Geraden* hat, welche durch `l` repräsentiert wird. Ist `l` nicht horizontal, dann kann sich Ihre Methode beliebig verhalten. In dieser Teilaufgabe interpretieren wir `l` also als unendlich lange Gerade, die durch die beiden Endpunkte des entsprechenden `LineSegment`s aufgespannt wird. Schreiben Sie außerdem eine weitere Methode `boolean intersectHorizontal(LineSegment[] parallel)`. Diese soll genau dann `true` zurückgeben, wenn `this` mit mindestens einer der horizontalen Geraden aus `parallel` einen Schnittpunkt hat. Falls `parallel` auch nicht-horizontale Strecken enthält, darf sich Ihre Methode wieder beliebig verhalten.

g) Schreiben Sie in der Klasse `LineSegment` die Methode `double computeValue(LineSegment[] parallel, LineSegment[] random)`. Diese soll für ein Array `random` von n (also $n =$ `random.length`) zufälligen Strecken, die Anzahl m der Strecken berechnen, die mit mindestens einer der horizontalen Geraden aus `parallel` einen Schnittpunkt haben. Geben Sie den Wert $2 \cdot \frac{n}{m}$ zurück. Existiert kein Schnittpunkt, dann geben Sie `0.0` zurück. Kompilieren Sie nun die Datei `Plot.java` und führen Sie die `main`-Methode der Klasse `Plot` aus. Die Klasse `Plot` stellt eine einfache Oberfläche (siehe unten) zur Verfügung und verwendet die von Ihnen zuvor geschriebenen Klassen. Daher ist es wichtig, dass die Methoden- und Klassennamen nicht verändert wurden. In der Oberfläche werden 13 parallele Strecken (mit Abstand von 50 (= 600/12) Einheiten) in einem 600×600 Einheiten großen Feld gezeichnet. Hierbei hat die linke obere Ecke die Koordinate `(0,0)` und die Ecke unten links `(0,599)`. Mit den jeweiligen Buttons können Sie nun zufällige Strecken mit Länge 50 Einheiten in diesem Feld erzeugen und zeichnen. Fällt Ihnen etwas beim ausgegebenen Wert auf?

Hinweise:

- Die drei Dateien `Point.java`, `LineSegment.java` und `Plot.java` müssen sich im selben Ordner befinden. Kompilieren Sie dann mit `javac Plot.java` und führen Sie danach das Programm mit `java Plot` aus.

h) Schreiben Sie ausführliche `javadoc`-Kommentare für beide Klassen `Point` und `LineSegment`.

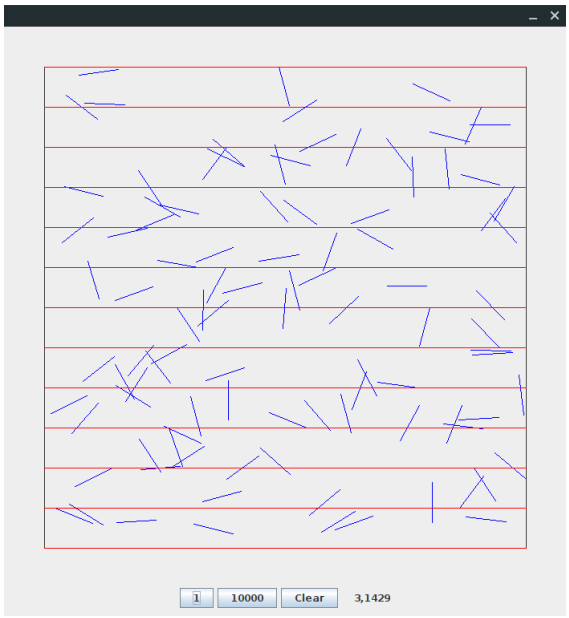

Abbildung 1: Oberfläche

Hausaufgabe 3 (Programmierung mit Datenabstraktion): (30 Punkte)

In dieser Aufgabe implementieren Sie eines der bedeutendsten kryptographischen Verfahren der Neuzeit: das RSA-Verfahren[1]. Die Erfinder Ronald L. Rivest, Adi Shamir und Leonard Adleman erhielten hierfür unter anderem 2002 den Turing Award. Hierzu verwenden Sie in dieser Aufgabe die Klasse `BigInteger`, die es ermöglicht, mit beliebig großen Zahlen ohne Überläufe zu rechnen. Es ist in der Kryptographie besonders wichtig mit großen Zahlen zu rechnen, damit der Geheimtext nicht durch bloßes Ausprobieren entschlüsselt werden kann.

`https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/math/BigInteger.html`

Damit die Klasse verwendet werden kann, müssen Ihre Dateien mit

```
import java.math.BigInteger;
```

beginnen.
RSA ist ein asymmetrisches kryptographisches Verfahren, d.h., es gibt einen öffentlichen Schlüssel (n, d) und einen privaten Schlüssel (n, e) von Zahlen d, e und n. Erfüllen die Schlüssel gewisse mathematische Eigenschaften, welche wir hier nicht weiter spezifizieren, dann kann man eine "Nachrichtenzahl" m verschlüsseln als $c = m^d \bmod n$. Die verschlüsselte Nachricht c kann dann mit $m = c^e \bmod n$ wieder entschlüsselt werden. Beachten Sie in allen Teilaufgaben die *Prinzipien der Datenkapselung*.

a) Schreiben Sie eine Klasse `RSA`. Diese Klasse hat drei Attribute `d`, `e` und `n` vom Typ `BigInteger`. In den folgenden Teilaufgaben ergänzen Sie diese Klasse.

i) Die Eulersche Phi-Funktion φ sei definiert durch $\varphi(p, q) = (p-1)\cdot(q-1)$. Schreiben Sie eine statische Methode `BigInteger eulerPhi(BigInteger p, BigInteger q)`, welche $\varphi(\mathtt{p}, \mathtt{q})$ berechnet für die beiden Primzahlen `p` und `q` vom Typ `BigInteger`. Zum Beispiel erhält man für $p = 53503$ und $q = 91957$ das Ergebnis $\varphi(53503, 91957) = 4919829912$.

[1] `https://de.wikipedia.org/wiki/RSA-Kryptosystem`

Hinweise:

- In dieser Aufgabe können Sie immer davon ausgehen, dass `p` und `q` Primzahlen sind.
- Die Klasse `BigInteger` stellt die Methoden `subtract` und `multiply` und eine Konstante `ONE` zur Verfügung. Lesen Sie für detaillierte Beschreibungen die Erklärungen in der verlinkten Java-Dokumentation sorgfältig durch.

ii) Schreiben Sie eine statische Methode `boolean lessThan(BigInteger a, BigInteger b)`, welche genau dann `true` zurückgibt, wenn `a` kleiner ist als `b`. Sie können hierzu die Methode `compareTo` aus der Klasse `BigInteger` verwenden. Hierbei ist `a.compareTo(b)` genau dann negativ, wenn `a` kleiner ist als `b`.

iii) Schreiben Sie eine statische Methode `BigInteger generateCoprime(BigInteger x)`, die zu einer gegebenen positiven Zahl `x` eine zufällige Zahl `e` generiert, welche die folgenden Bedingungen erfüllt:

- `e` liegt strikt zwischen 1 und `x`, also $1 < \texttt{e} < \texttt{x}$. (Ist $\texttt{x} \leq 1$, dann kann sich Ihre Methode beliebig verhalten.)
- `e` ist teilerfremd zu `x`, d.h. der größte gemeinsame Teiler von `e` und `x` ist $\gcd(\texttt{e}, \texttt{x}) = 1$.

Die Methode soll wiederholt zufällige Kandidaten `e` mittels `new BigInteger(x.bitLength(), new Random())` erzeugen, bis ein Wert gefunden wird, der beide Bedingungen erfüllt. Zum Beispiel erfüllt die Zahl 2678923603 beide Bedingungen, wenn man `x` als $\varphi(53503, 91957) = 4919829912$ wählt.

Hinweise:

- Sie müssen die Klasse `Random` mittels `import java.util.Random;` am Anfang Ihrer Datei importieren.
- Die Klasse `BigInteger` stellt eine Methode `gcd` und eine Methode `equals` zur Verfügung.

b) Implementieren Sie nun zwei Konstruktoren für die Klasse `RSA`, welche die Attribute `d`, `e` und `n` setzen:

i) `public RSA(long p, long q)`

ii) `public RSA(BigInteger p, BigInteger q)`

Hierbei soll `n` das Produkt der Primzahlen `p` und `q` sein. Die Zahl `e` soll eine beliebige teilerfremde Zahl zu $\varphi(\texttt{p}, \texttt{q})$ sein. Schließlich soll `d` die Zahl sein, sodass $\texttt{d} \cdot \texttt{e} \bmod \varphi(\texttt{p}, \texttt{q}) = 1$ gilt. Die Zahl `d` ist also das Inverse zur Zahl `e` modulo $\varphi(\texttt{p}, \texttt{q})$. Die Methode `modInverse` der Klasse `BigInteger` berechnet ein solches Inverses. Für das obige Beispiel erhält man $\texttt{d} = 1934167555$, $\texttt{e} = 2678923603$ und $\texttt{n} = 4919975371$.

c) Schreiben Sie zwei `record`-Klassen `PublicKey` und `PrivateKey`, welche den öffentlichen bzw. privaten Schlüssel repräsentieren. Die Klasse `PublicKey` hat die beiden Attribute `n` und `d`. Die Klasse `PrivateKey` hat die beiden Attribute `n` und `e`. Implementieren Sie nun die Methode `BigInteger encrypt(BigInteger message)` in der Klasse `PublicKey`, die die Zahl `message` verschlüsselt. Implementieren Sie außerdem die Methode `BigInteger decrypt(BigInteger cipher)` in der Klasse `PrivateKey`, die die Zahl `cipher` entschlüsselt. Zum Beispiel wird mit obigen Werten von `d` und `n` die Zahl 100000 zu $100000^{\texttt{d}} \bmod \texttt{n} = 1448759588$ verschlüsselt und kann mit `e` wieder durch $1448759588^{\texttt{e}} \bmod \texttt{n} = 100000$ entschlüsselt werden.

Hinweise:

- Sie können die Methode `modPow` der Klasse `BigInteger` verwenden, um `BigInteger` zu potenzieren. Hierbei entspricht `a.modPow(b,c)` der Berechnung $\texttt{a}^{\texttt{b}} \bmod \texttt{c}$.

d) Erweitern Sie die Klasse `RSA` nun um zwei neue Methoden `PublicKey publicKey()` und `PrivateKey privateKey()`, welche für ein konkretes `RSA`-Objekt den öffentlichen bzw. privaten Schlüssel zurückgeben.

Hinweise:

- Im Moodle-Lernraum steht eine Klasse `RSAMain` zur Verfügung, mit welcher Sie Ihre Implementierung testen können. Hier werden die beiden Primzahlen $p = 53503$ und $q = 91957$ verwendet und die Nachricht 100000 erst verschlüsselt und danach wieder entschlüsselt. Beachten Sie, dass `e` zufällig gewählt wird. Damit verändert sich also in jeder Ausführung die Verschlüsselung. Die entschlüsselte Nachricht sollte aber immer 100000 sein.

e) Schreiben Sie ausführliche `javadoc`-Kommentare für alle drei Klassen `RSA`, `PublicKey` und `PrivateKey`.

Übungsaufgabe 4 (Programmanalyse):

Lösen Sie die folgende Aufgabe ohne Einsatz eines Computers. Bedenken Sie, dass Sie in einer Prüfungssituation ebenfalls keinen Computer zur Verfügung haben.
Betrachten Sie das folgende kurze Programm:

```
public class A {
    public static void main() {
        A a = new A();

        a.a(Long.valueOf(100));         //a)
        a.a(Double.valueOf(100));       //b)
        a.a(Integer.valueOf(100));      //c)

        b(Integer.valueOf(100), "0"); //d)
        b(100L, "0");                 //e)
        b(100L, '0');                 //f)
    }

    public void a(int p) {
        IO.println("a1");
    }

    public void a(double p) {
        IO.println("a2");
    }

    public void a(Double p) {
        IO.println("a3");
    }


    public static void b(Long p1, int p2) {
        IO.println("b1");
    }

    public static void b(long p1, String p2) {
        IO.println("b2");
    }

    public static void b(Long p1, String p2) {
        IO.println("b3");
    }
}
```

Geben Sie die Ausgabe dieses Programms an, wenn die `main`-Methode ausgeführt wird. **Begründen Sie Ihre Antwort!** Ordnen Sie jeder Teilaufgabe die aufgetretenen Effekte zu und erklären Sie, warum gerade diese zu beobachten sind.

Hausaufgabe 5 (Programmanalyse): (20 Punkte)

Lösen Sie die folgende Aufgabe ohne Einsatz eines Computers. Bedenken Sie, dass Sie in einer Prüfungssituation ebenfalls keinen Computer zur Verfügung haben.
Betrachten Sie das folgende kurze Programm:

```
public class A {
    private int i1;
    private Integer i2;
    private short s;

    public A(Short s, int i) {
        this.i1 = s;
        this.i2 = i;
        this.s = s;
    }

    public void f(Integer i) {
        IO.println("f1");
    }
    public void f(String s) {
        IO.println("f2: " + s);
    }
    public void f(double d) {
        IO.println("f3");
    }

    public void g(int i, float d) {
        IO.println("g1");
    }
    public void g(Integer i, Long j) {
        IO.println("g2");
    }
    public void g(int... is) {
        IO.println("g3");
    }
    public void g(double... is) {
        IO.println("g4");
    }

    public static void main() {
        A a = new A((short)1,2);
        a.f(a.i1);                    //a)
        a.f(a.i2);                    //b)
        a.f(a.s);                     //c)
        a.f(.0f);                     //d)

        a.g(1,2);                     //e)
        a.g(1,2L);                    //f)
        a.g(a.i2,Long.valueOf(2));    //g)
        a.g(1,2.0);                   //h)
        a.g(1,2.0f);                  //i)
    }
}
```

Geben Sie die Ausgabe dieses Programms an, wenn die `main`-Methode ausgeführt wird. **Begründen Sie Ihre Antwort!** Ordnen Sie jeder Teilaufgabe die aufgetretenen Effekte zu und erklären Sie, warum gerade diese zu beobachten sind.

Hausaufgabe 6 (Deck 5): (Codescape)

Lösen Sie die Missionen von Deck 5 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn sie Ihre Lösung vor der einheitlichen Codescape Deadline am Freitag, den 30.01.2026, um 23:59 Uhr abschicken.