Karlsruher Institut für Technologie

Programmieren Tutorium Nummer 2

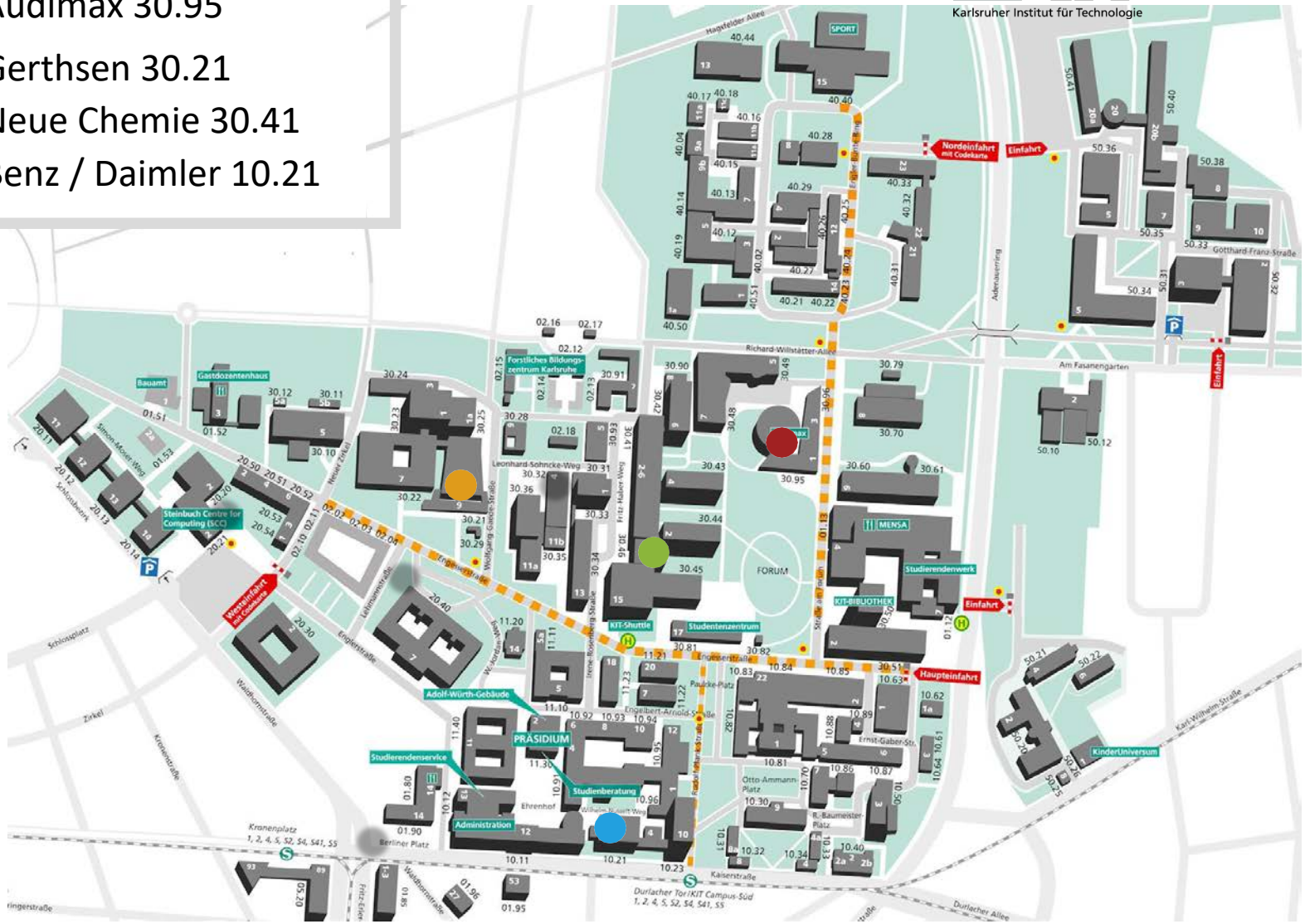
Präsenzübungsvorbereitung

Bild

Präsenzübung – Organisatorisches

- In drei Tagen am Donnerstag **17. Januar 2019** ab **17:15 Uhr**
- Hintereinander in **zwei Gruppen** in **fünf verschiedenen Hörsälen**
- An der Präsenzübung kann nur teilnehmen, wer sich bis zum 10. Januar 2019 für den **Übungsschein angemeldet** hatte
- Übungsblätter > 50% ∧ Präsenzübung > 75% ⇒ Übungsschein
- Übungsblätter ≤ 50% ∨ Präsenzübung ≤ 75% ⇒ Beides wiederholen
- Übungsblätter > 50% ∧ ärztliches Attest ⇒ Präsenzübung wiederholen
- Jeder **Betrugsversuch** zählt sofort als **„nicht bestanden“**

- Audimax 30.95
- Gerthsen 30.21
- Neue Chemie 30.41
- Benz / Daimler 10.21



Präsenzübung – Formalitäten

- **Sitzplan einhalten**
- **Studentenausweis** und **Personalausweis** am Platz
- Alle Aufgaben sind **ohne jegliche Hilfsmittel** zu lösen
  - Keine Spickzettel oder sonstige Aufzeichnungen
  - Keine Elektronik (Mobiltelefon, Smartwatch, Taschenrechner)
- Nur **dokumentenechte Stifte**, welche **nicht rot** sind
  - Zum Beispiel ein Kugelschreiber mit blauer Tinte
  - Aber kein Bleistift
- Die Aufgaben werden in der Präsenzübung **nur auf Deutsch** gestellt
  - Nicht **geprüfte Wörterbücher** dürfen nicht verwendet werden

Präsenzübung – Ablauf

- Aufgabenblätter werden verteilt
- **Erst öffnen, wenn es der Übungsleiter sagt**
- Den vollen **Namen**, **Matrikelnummer** und die **Nummer des Tutoriums (2)** auf die Aufgabenblätter schreiben
- Für eine korrekte Lösung müssen **nicht** unbedingt **alle Lücken** in einer Aufgabe ausgefüllt werden
- Die **Größe** der Lücken steht nicht unbedingt in Relation zur **Länge** oder der **Komplexität** der Lösung
- Es müssen keine Vorgaben zum **Programmierstil** eingehalten werden
- **20 Minuten** Bearbeitungszeit und mit der Abgabe warten, bis alles **eingesammelt** und **ausgezählt** wurde
- Maximal 14 Punkte

Grundlagen 1

- Implementieren Sie ein Programm, welches auf möglichst einfache Weise, den Text `Hallo Welt!` ausgibt.

Grundlagen 1

- Implementieren Sie ein Programm, welches auf möglichst einfache Weise, den Text `Hallo Welt!` ausgibt.

```
public class HelloWorld {

        public static void main(String[] args) {

                System.out.println("Hallo Welt!");

        }

}
```

Grundlagen 2

- Implementieren Sie ein Programm, welches den Wert zweier `int`-Variablen `a` und `b` vertauscht:

```
public class Swap {
    public static void main(String[] args) {
        int a = 2;
        int b = 4;
```



```
        System.out.println("a = " + a + ",  b = " + b);
    }
}
```

Grundlagen 2

- Implementieren Sie ein Programm, welches den Wert zweier `int`-Variablen a und b vertauscht:

```
public class Swap {
    public static void main(String[] args) {
        int a = 2;
        int b = 4;

        int h = a;
        a = b;
        b = h;

        System.out.println("a = " + a + ",  b = " + b);
    }
}
```

Grundlagen 3

- Vervollständigen Sie die Anweisung, sodass es nicht zu einer `NullPointerException` kommt, falls der Parameter `s` gleich `null` sein sollte.

```
public static final boolean isBig(final String s) {
    return                          .length() > 100;
}
```

Grundlagen 3

- Vervollständigen Sie die Anweisung, sodass es nicht zu einer `NullPointerException` kommt, falls der Parameter `s` gleich `null` sein sollte.

```
public static final boolean isBig(final String s) {

    return      s != null      && s.length() > 100;

}
```

Grundlagen 4

- Vervollständigen Sie die Lücke, sodass wenn das Array mindestens zwei Elemente enthält, das zweite Element zurückgeben wird. Falls dies nicht der Fall ist, wird `-1` zurückgegeben.

```
public final int getSecond(int[] array) {
        if(                                    ) {




        }

}
```

Grundlagen 4

- Vervollständigen Sie die Lücke, sodass wenn das Array mindestens zwei Elemente enthält, das zweite Element zurückgeben wird. Falls dies nicht der Fall ist, wird `-1` zurückgegeben.

```
public final int getSecond(int[] array) {
        if(array != null && array.length >= 2) {

                        return array[1];

        }
        return -1;
}
```

Grundlagen 5

- Vervollständigen Sie die Methode, welche nur dann `true` zurück gibt, wenn man ausschlafen kann. Der Parameter `weekday` ist nur dann `true`, wenn es ein Wochentag ist, und der Parameter `vacation` ist nur dann `true`, wenn man im Urlaub ist. Mann kann nur dann ausschlafen, wenn es kein Wochentag ist oder man im Urlaub ist.

```
public boolean sleepIn(boolean weekday, boolean vacation) {

}
```

Grundlagen 5

- Vervollständigen Sie die Methode, welche nur dann `true` zurück gibt, wenn man ausschlafen kann. Der Parameter `weekday` ist nur dann `true`, wenn es ein Wochentag ist, und der Parameter `vacation` ist nur dann `true`, wenn man im Urlaub ist. Mann kann nur dann ausschlafen, wenn es kein Wochentag ist oder man im Urlaub ist.

```
public boolean sleepIn(boolean weekday, boolean vacation) {

        return !weekday || vacation;

}
```

Grundlagen 6

- Vervollständigen Sie die Methode, welche nur dann `true` zurück gibt, wenn mindestens eine der drei übergebenen Zahlen valide ist. Eine Zahl ist nur dann valide, wenn sie sich in dem Bereich von einschließlich 4 bis einschließlich 8 befindet.

```
public boolean hasValide(int a, int b, int c) {

}
```

Grundlagen 6

- Vervollständigen Sie die Methode, welche nur dann `true` zurück gibt, wenn mindestens eine der drei übergebenen Zahlen valide ist. Eine Zahl ist nur dann valide, wenn sie sich in dem Bereich von einschließlich 4 bis einschließlich 8 befindet.

```
public boolean hasValide(int a, int b, int c) {


    return          (a >= 4 && a <= 8) ||
                           (b >= 4 && b <= 8) ||
                    (c >= 4 && c <= 8);


}
```

Grundlagen 7

- Vervollständigen Sie die Methode, welche nur dann `true` zurück gibt, wenn die gegebene nicht negative Zahl `x` ein Vielfaches von `3` oder ein Vielfaches von `5` ist.

```
public boolean isDivisible35(final int x) {

}
```

Grundlagen 7

- Vervollständigen Sie die Methode, welche nur dann `true` zurück gibt, wenn die gegebene nicht negative Zahl `x` ein Vielfaches von `3` oder ein Vielfaches von `5` ist.

```
public boolean isDivisible35(final int x) {

                return (x % 3 == 0) || (x % 5 == 0);

}
```

Grundlagen 8

- Implementieren Sie in der Klasse `Main` eine statische Methode mit dem Bezeichner `neg` und einem Rückgabewert des Typs `boolean`. Diese Methode nimmt als einzigen Parameter eine boolesche Variable mit dem Bezeichner `b` entgegen. Wenn diese Variable `true` ist, dann wird `false` zurückgeben und wenn diese Variable `false` ist, dann wird `true` zurückgeben.

```
public class Main {

}
```

Grundlagen 8

- Implementieren Sie in der Klasse `Main` eine statische Methode mit dem Bezeichner `neg` und einem Rückgabewert des Typs `boolean`. Diese Methode nimmt als einzigen Parameter eine boolesche Variable mit dem Bezeichner `b` entgegen. Wenn diese Variable `true` ist, dann wird `false` zurückgeben und wenn diese Variable `false` ist, dann wird `true` zurückgeben.

```
public class Main {


        static boolean neg(boolean b) {
                        return !b;
                }


}
```

Grundlagen 9

- Implementieren Sie diese Methode, sodass diese ein `int`-Array mit nur den ersten drei Stellen der Kreiszahl Pi `(3, 1, 4)` zurückgibt.

```
public static int[] pi() {

}
```

Grundlagen 9

- Implementieren Sie diese Methode, sodass diese ein `int`-Array mit nur den ersten drei Stellen der Kreiszahl Pi `(3, 1, 4)` zurückgibt.

```
public static int[] pi() {

        return new int[] { 3, 1, 4 };

}
```

Grundlagen 10

- Vervollständigen Sie die Methode `close10`, welche denn Wert wieder zurück gibt, welcher am nächsten an der Zahl `10` ist. Falls beide übergebene Werte gleich nah dran sind, soll `0` zurück geben werden.
- Hinweis: `Math.abs(n)` gibt den Betrag einer Zahl zurück.

```
public static int close10(int a, int b) {
                    return 0;
}
```

Grundlagen 10

- Vervollständigen Sie die Methode `close10`, welche denn Wert wieder zurück gibt, welcher am nächsten an der Zahl `10` ist. Falls beide übergebene Werte gleich nah dran sind, soll `0` zurück geben werden.
- Hinweis: `Math.abs(n)` gibt den Betrag einer Zahl zurück.

```
public static int close10(int a, int b) {
        int aDiff = Math.abs(a - 10);
        int bDiff = Math.abs(b - 10);
        if (aDiff < bDiff) {
                        return a;
                } else if (bDiff < aDiff) {
                        return b;
                }
        return 0;
}
```

Kontrollfluss 1

- Vervollständigen Sie folgende Methode, welcher ein instanziierte Zeichenfolge `str` und eine natürliche Zahl `n` übergeben werden. Die Methode soll die ursprüngliche Zeichenfolge `n`-mal hintereinander kopieren und diese neue Zeichenfolge zurückgeben.
- Beispiel: `stringTimes("Hi", 3) == "HiHiHi"`

```
public static String stringTimes(String str, int n) {

}
```

Kontrollfluss 1

- Vervollständigen Sie folgende Methode, welcher ein instanziierte Zeichenfolge `str` und eine natürliche Zahl `n` übergeben werden. Die Methode soll die ursprüngliche Zeichenfolge `n`-mal hintereinander kopieren und diese neue Zeichenfolge zurückgeben.
- Beispiel: `stringTimes("Hi", 3) == "HiHiHi"`

```
public static String stringTimes(String str, int n) {

        String result = "";
        for (int i = 0; i < n; i++) {
                        result += str;
        }
        return result;

}
```

Kontrollfluss 2

- Vervollständigen Sie folgende Methode, welche überprüft ob die Zahlenfolge `1, 2, 3` in einem Array vorkommt.
- Beispiel: `has123({4,1,2,3,0}) == true`

```
public boolean has123(int[] array) {
        for (
                        if (array[i] == 1 &&
                                array[i + 1] == 2 &&
                            array[i + 2] == 3) {
                                return true;
                        }
                }
        return false;
}
```

Kontrollfluss 2

- Vervollständigen Sie folgende Methode, welche überprüft ob die Zahlenfolge `1, 2, 3` in einem Array vorkommt.
- Beispiel: `has123({4,1,2,3,0}) == true`

```
public boolean has123(int[] array) {

        for (   int i = 0; i < (array.length - 2); i++    ) {

                        if (array[i] == 1 &&
                                    array[i + 1] == 2 &&
                                array[i + 2] == 3) {
                                        return true;
                        }
                }
        return false;
}
```

Kontrollfluss 3

- Implementieren Sie die Methode `swap`, sodass die Reihenfolge der Inhalte im dem übergebenen Array umgekehrt werden.
- Beispiel: `swap({1, 2, 3}) == {3, 2, 1}`

```
public static int[] swap(int[] array) {
        int length = array.length;
        for (int i = 0; i < length / 2; i++) {
                        int tmp = array[          ];
                        array[          ] = array[          ];
                        array[          ] = tmp;
        }
        return array;
}
```

Kontrollfluss 3

- Implementieren Sie die Methode `swap`, sodass die Reihenfolge der Inhalte im dem übergebenen Array umgekehrt werden.
- Beispiel: `swap({1, 2, 3}) == {3, 2, 1}`

```
public static int[] swap(int[] array) {
        int length = array.length;
        for (int i = 0; i < length / 2; i++) {
                        int tmp = array[      i      ];
                        array[     i     ] = array[ length - i - 1 ];
                        array[ length - i - 1 ] = tmp;
        }
        return array;
}
```

Kontrollfluss 4

- Vervollständigen Sie die Lücken, sodass die Methode `sum` für ein instanziiertes nichtleeres Array alle beinhalteten Zahlenwerte rekursiv aufaddiert und diesen Wert zurückgibt.

```
public class Summation {
        public static int sum(int[] array) {
                        return sum(                                    );
        }
        public static int sum(int[] array, int i) {
                        if (                        ) {
                                        return                      ;
                        }
                        return array[i] + sum(                         );
                }
}
```

Kontrollfluss 4

- Vervollständigen Sie die Lücken, sodass die Methode `sum` für ein instanziiertes nichtleeres Array alle beinhalteten Zahlenwerte rekursiv aufaddiert und diesen Wert zurückgibt.

```
public class Summation {
        public static int sum(int[] array) {
                        return sum( array,      array.length - 1      );
        }
        public static int sum(int[] array, int i) {
                        if (         i < 0          ) {
                                        return            0           ;
                        }
                        return array[i] + sum(array,      i - 1       );
                }
}
```

Kontrollfluss 5

- Implementieren Sie die Methode `c4` aus, sodass diese zurückgibt, wie oft die Zahl `4` in dem ihr übergeben Array enthalten ist.
- Beispiel: `c4({4,1,2,3,4}) == 2`

```
public static int c4(final int[] array) {
        
        
        
    
}
```

Kontrollfluss 5

- Implementieren Sie die Methode `c4` aus, sodass diese zurückgibt, wie oft die Zahl 4 in dem ihr übergeben Array enthalten ist.
- Beispiel: `c4({4,1,2,3,4}) == 2`

```
public static int c4(final int[] array) {
        int count = 0;
        for (final int element : array) {
                    if (element == 4) {
                            count++;
                    }
                }
        return count;
}
```

Kontrollfluss 6

- Vervollständigen Sie die Lücken, sodass die Methode `count` beginnend bei einschließlich `0` bis einschlich zu der ihr übergebenen natürlichen Zahl `to` um `1` hochhält und jeden Schritt ausgibt. Beispiel: `count(3)` gibt jeweils `0 1 2 3` in einer eigenen Zeile aus.

```
public static void count(int to) {
        int i = 
        while ( ) {
                
                
                
        }
    
}
```

Kontrollfluss 6

- Vervollständigen Sie die Lücken, sodass die Methode `count` beginnend bei einschließlich `0` bis einschlich zu der ihr übergebenen natürlichen Zahl `to` um `1` hochhält und jeden Schritt ausgibt. Beispiel: `count(3)` gibt jeweils `0 1 2 3` in einer eigenen Zeile aus.

```
public static void count(int to) {
        int i = 0;
        while (          i <= to          ) {

                        System.out.println(          i++          );
        }

}
```

Kontrollfluss 7

- Deklarieren Sie die Variable `end`, sodass diese Methode `true` zurückgibt, wenn eines der ersten `4` Elemente im Array ein `8` ist.

```
public static boolean front8(int[] array) {
        int end =
        for (int i = 0; i < end; i++) {
                    if (array[i] == 8) {
                                return true;
                    }
        }
        return false;
}
```

Kontrollfluss 7

- Deklarieren Sie die Variable `end`, sodass diese Methode `true` zurückgibt, wenn eines der ersten 4 Elemente im Array ein 8 ist.

```
public static boolean front8(int[] array) {
        int end = array.length;
        if (end > 4) {
                        end = 4;
        }

        for (int i = 0; i < end; i++) {
                        if (array[i] == 8) {
                                        return true;
                                }
        }
        return false;
}
```

Kontrollfluss 8

- Vervollständigen Sie die Methode `max`, welche den größten Wert der von den drei übergeben Werte wieder zurück gibt.

```
public static int max(int a, int b, int c) {

}
```

Kontrollfluss 8

- Vervollständigen Sie die Methode `max`, welche den größten Wert der von den drei übergeben Werte wieder zurück gibt.

```
public static int max(int a, int b, int c) {
        int max;
                if (a > b) {
                        max = a;
                } else {
                        max = b;
                }
        if (c > max) {
                        max = c;
                }
                return max;
}
```

Kontrollfluss 9

- Vervollständigen Sie die Lücken, sodass die Methode rekursiv die `n`-te Potenz von `x` berechnet. Sie könne davon ausgehen, dass die beiden Parameter `x` und `n`, bei ersten Aufruf größer `0` sind.

```
int power(int x, int n) {
        if (                                  ) {
                                                  ;
        }
        return x * power(x,                       );
}
```

Kontrollfluss 9

- Vervollständigen Sie die Lücken, sodass die Methode rekursiv die `n`-te Potenz von `x` berechnet. Sie könne davon ausgehen, dass die beiden Parameter `x` und `n`, bei ersten Aufruf größer `0` sind.

```
int power(int x, int n) {
        if (            n <= 0            ) {
                      return          1             ;
        }
        return x * power(x,           n - 1           );
}
```

Kontrollfluss 10

- Vervollständigen Sie die Lücken, sodass die Variable `element` bei jedem Schleifendurchlauf einen neuen Wert der Matrix zugeordnet wird, sodass alle Werte ausgegeben werden. Gehen Sie davon aus, dass der Parameter `matrix` vollständig instanziiert und nicht leer ist.

```
public static void print(double[][] matrix) {
        for (                                    ) {
                for (                                    ) {
                        double element =     matrix[i][j]     ;
                        System.out.print(element + " ");
                }
        }
}
```

Kontrollfluss 10

- Vervollständigen Sie die Lücken, sodass die Variable `element` bei jedem Schleifendurchlauf einen neuen Wert der Matrix zugeordnet wird, sodass alle Werte ausgegeben werden. Gehen Sie davon aus, dass der Parameter `matrix` vollständig instanziiert und nicht leer ist.

```
public static void print(double[][] matrix) {

        for ( int i = 0; i < matrix.length; i++ ) {

                for ( int j = 0; j < matrix[i].length; j++ ) {

                        double element =     matrix[i][j]     ;
                        System.out.print(element + " ");
                }
                System.out.println();
        }
}
```

Objektorientierung 1

- Implementieren Sie eine Klasse, welche lediglich eine ganzen Zahlen und ein Gleitkommazahl als Attribut beinhaltet.

Objektorientierung 1

- Implementieren Sie eine Klasse, welche lediglich eine ganzen Zahlen und ein Gleitkommazahl als Attribut beinhaltet.

```
class MyClass {

        int integer;
        double floatingPoint;

}
```

Objektorientierung 2

- Implementieren Sie eine Aufzählungstyp (`enum`) mit dem Bezeichner `Direction`, welcher die vier Himmelsrichtung `N`, `E`, `S` und `W` definiert.

Objektorientierung 2

- Implementieren Sie eine Aufzählungstyp (`enum`) mit dem Bezeichner `Direction`, welcher die vier Himmelsrichtung `N`, `E`, `S` und `W` definiert.

```
enum Direction{N, E, S, W}
```

Objektorientierung 3

- Vervollständigen Sie den Konstruktor, sodass alle Attribute mit den Parametern initialisiert werden.

```
public class Container {
        private final int value;
        private final String unit;

        public Container(int value, String unit) {




        }
}
```

Objektorientierung 3

- Vervollständigen Sie den Konstruktor, sodass alle Attribute mit den Parametern initialisiert werden.

```
public class Container {
        private final int value;
        private final String unit;

        public Container(int value, String unit) {

                        this.value = value;
                        this.unit = unit;

        }
}
```

Objektorientierung 4

- Vervollständigen Sie die Methode, sodass diese abhängig von den Werten der Attribute einen String nach dem Muster „`(x, y)`“ zurückgibt.
- Beispielsweise: `(-4, 2)` oder `(13, 37)`

```
class Point {
        int x;
        int y;

        }
}
```

Objektorientierung 4

- Vervollständigen Sie die Methode, sodass diese abhängig von den Werten der Attribute einen String nach dem Muster „`(x, y)`“ zurückgibt.
- Beispielsweise: `(-4, 2)` oder `(13, 37)`

```
class Point {
        int x;
        int y;

        @Override
        public String toString() {

                        return "(" + x + ", " + y + ")";

        }
}
```

Objektorientierung 6

- Fügen Sie den zwei Attributen alle zugehörigen möglichen und sinnvollen Zugriffsmethoden (getter/setter) hinzu.

```
public class Container {
        private final int i;
        private float f;
        public Container(int i, float f) {
                        this.i = i;
                        this.f = f;
        }
}
```

Objektorientierung 6

- Fügen Sie den zwei Attributen alle zugehörigen möglichen und sinnvollen Zugriffsmethoden (getter/setter) hinzu.

```
public class Container {
        private final int i;
        private float f;
        public Container(int i, float f) {
                        this.i = i;
                        this.f = f;
        }

        public int getI() { return i; }

        public float getF() { return f; }

        public void setF(float f) { this.f = f; }
}
```

Objektorientierung 7

- Implementieren sie eine Klasse, mit welcher mehrere Boolesche Variablen miteinander verketten werden können. Das heißt die Klasse beinhaltet eine Boolesche Variable und die Referenz auf eine weitere Boolesche Variable, welche wieder eine Referenz auf eine weitere Boolesche Variablen hat und so weiter.

Objektorientierung 7

- Implementieren sie eine Klasse, mit welcher mehrere Boolesche Variablen miteinander verketten werden können. Das heißt die Klasse beinhaltet eine Boolesche Variable und die Referenz auf eine weitere Boolesche Variable, welche wieder eine Referenz auf eine weitere Boolesche Variablen hat und so weiter.

```
class Container {
        boolean b;
        Container c;
}
```

Objektorientierung 8

- Falls die Zeile nicht zu einem Fehler führt, was ist dann ihr Rückgabewert?

```
abstract class Parent {
  String foo(){return "pf";}
}

class Child extends Parent {
  String foo(){return "cf";}
  String bar(){return "cb";}
}

class Grand extends Child {
  String foo(){return "gf";}
}
```

```
Parent p = new Child();
p.foo();
p.bar();
p = new GrandChild();
p.foo();
p.bar();
Child c = new Child();
c.foo();
c.bar();
c = new GrandChild();
c.foo();
c.bar();
```

Objektorientierung 8

- Falls die Zeile nicht zu einem Fehler führt, was ist dann ihr Rückgabewert?

```
abstract class Parent {
  String foo(){return "pf";}
}

class Child extends Parent {
  String foo(){return "cf";}
  String bar(){return "cb";}
}

class Grand extends Child {
  String foo(){return "gf";}
}
```

```
Parent p = new Child();
p.foo();        cf
p.bar();        X
p = new GrandChild();
p.foo();        gf
p.bar();        X
Child c = new Child();
c.foo();        cf
c.bar();        cb
c = new GrandChild();
c.foo();        gf
c.bar();        cb
```

Objektorientierung 9

- Schreiben Sie Java-Code für eine Schnittstelle, die eine Methodensignatur Ihrer Wahl enthält und eine Klasse, die diese Schnittstelle implementiert. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

Objektorientierung 9

- Schreiben Sie Java-Code für eine Schnittstelle, die eine Methodensignatur Ihrer Wahl enthält und eine Klasse, die diese Schnittstelle implementiert. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

```
public interface Worker {
	void work();
}

public class HardWorker implements Worker {
	public void work() { }
}
```

Objektorientierung 10

- Schreiben Sie Java-Code für eine Klasse, die eine abstrakte Methodensignatur Ihrer Wahl enthält und eine Klasse, die von dieser Klasse erbt und die Methode überschreibt. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

Objektorientierung 10

- Schreiben Sie Java-Code für eine Klasse, die eine abstrakte Methodensignatur Ihrer Wahl enthält und eine Klasse, die von dieser Klasse erbt und die Methode überschreibt. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

```
abstract class Worker {
        abstract void work();
}

class HardWorker extends Worker {
        @Override
        void work() { }
}
```

- Vielen Dank für eure Aufmerksamkeit
- Viel Erfolg beim Übungsblatt und der Präsenzübung