

G



D



D

# 1 Einstieg in die Programmierung

## 1.1 Theoretische Grundlagen von Programmiersprachen

Bevor Sie sich konkret mit einer Programmiersprache beschäftigen, sollen Sie sich erstmal über verschiedene Eigenschaften von Programmiersprachen informieren.

### Aufgabe:

1. Bearbeiten Sie in Ihrer Gruppe die zugewiesene Aufgabe. Im Anschluss werden Sie mit Mitgliedern der anderen Gruppen gemischt und sollen dort Ihre Ergebnisse den anderen erklären.

### Gruppe A: Arten von Programmiersprachen

Erklären Sie kurz den Ansatz folgender Programmiersprachen und nennen Sie für jede Sprache konkrete Beispiele.

- Strukturierte Programmiersprachen

#### 1. Strukturierte Programmiersprachen:

- **Ansatz:** Strukturierte Programmiersprachen verwenden klare Strukturen, Kontrollflussanweisungen und Prozeduren, um den Programmfluss zu steuern.
- **Beispiele:** C, Pascal

#### 2. Imperative Programmiersprachen:

- **Ansatz:** Imperative Programmiersprachen betonen die Veränderung des Programmzustands durch die Ausführung von Anweisungen. Hierbei stehen Anweisungen im Vordergrund.
- **Beispiele:** Fortran, C

#### 3. Deklarative Programmiersprachen:

- **Ansatz:** Deklarative Programmiersprachen legen den Fokus darauf, was erreicht werden soll, ohne explizit anzugeben, wie dies erreicht werden soll. Hierzu gehören funktionale und logische Programmiersprachen.
- **Beispiele:** SQL (für Datenbankabfragen), Prolog (für logische Programmierung)

#### 4. Objektorientierte Programmiersprachen:

- **Ansatz:** Objektorientierte Programmiersprachen verwenden Objekte, die Daten und Methoden kombinieren, um die Struktur und das Verhalten von Software zu modellieren.
- **Beispiele:** Java, C++, Python

#### 5. Funktionale Programmiersprachen:

- **Ansatz:** Funktionale Programmiersprachen behandeln Berechnungen als Auswertung von Funktionen und vermeiden Zustandsänderungen.
- **Beispiele:** Haskell, Lisp, Erlang

- Funktionale Programmiersprachen

## Gruppe B: Implementierungsformen von Software

Erklären Sie kurz (incl. Vor- bzw. Nachteilen) die folgenden Implementierungsformen von Software und nennen Sie für jede Implementierungsform typische Programmiersprachen als Beispiel.

### 1. Compiler:

- **Erklärung:** Compiler übersetzen den gesamten Quellcode einer Programmiersprache in Maschinencode, bevor das Programm ausgeführt wird.
- **Vorteile:** Effiziente Ausführung, da der Code bereits übersetzt wurde.
- **Nachteile:** Längere Entwicklungszeiten, da der gesamte Code übersetzt werden muss.
- **Beispiel:** C, C++

### 2. Interpreter:

- **Erklärung:** Interpreter übersetzen den Quellcode Zeile für Zeile und führen ihn gleichzeitig aus.
- **Vorteile:** Schnellere Entwicklungsszyklen, da nur der aktuelle Codeblock interpretiert wird.
- **Nachteile:** Möglicherweise geringere Laufzeitleistung im Vergleich zu kompilierten Sprachen.
- **Beispiel:** Python, Ruby

### 3. JIT-Compiler (Just-In-Time Compiler):

- **Erklärung:** JIT-Compiler kombinieren Aspekte von Compilern und Interpretern. Sie übersetzen den Code während der Laufzeit in Maschinencode.
- **Vorteile:** Kombiniert schnelle Ausführung mit schnellen Entwicklungsszyklen.
- **Nachteile:** Etwas höherer Overhead zur Laufzeit im Vergleich zu rein kompilierten Sprachen.
- **Beispiel:** Java (mit dem HotSpot JIT-Compiler)

## Gruppe C: Typisierung von Variablen

Erklären Sie kurz die unterschiedliche Arten von Variablen-Typisierungen und nennen Sie für jede Art eine passende Programmiersprache.

### Gruppe C: Typisierung von Variablen

#### 1. Starke vs. Schwache Typisierung:

- **Starke Typisierung:** Die Typen von Variablen werden strikt durch die Programmiersprache durchgesetzt.
- **Schwache Typisierung:** Die Programmiersprache erlaubt mehr Flexibilität in der Konvertierung von Typen.
- **Beispiel (starke Typisierung):** Java
- **Beispiel (schwache Typisierung):** JavaScript

#### 2. Dynamische vs. Statische Typisierung:

- **Dynamische Typisierung:** Variablentypen werden zur Laufzeit überprüft.
- **Statische Typisierung:** Variablentypen werden bereits zur Compilezeit überprüft.
- **Beispiel (dynamische Typisierung):** Python
- **Beispiel (statische Typisierung):** C

#### 3. Explizite vs. Implizite Typisierung:

- **Explizite Typisierung:** Der Entwickler muss den Typ einer Variable explizit angeben.
- **Implizite Typisierung:** Der Typ einer Variable wird automatisch vom System abgeleitet.
- **Beispiel (explizite Typisierung):** C#
- **Beispiel (implizite Typisierung):** Python (z.B., durch Verwendung von auto in modernen C++-Compilern)

# Gruppe C:

## Typisierung von Variablen

### stark vs schwach

stark: strenge Typisierung, erfordert explizite Def.  
 (z.B.: `int num = 5;`) Java

schwach: flexible Typisierung, erlaubt implizite Konvertierung (`let x = 5; x = "Hallo"`) JS

### dynamisch vs statisch

Dynamisch: Typ kann sich zur Laufzeit ändern  
`(x = 5, x = "Hallo")` Py

Statisch: Typ während der Kompilierung festlegen  
`(int num = 10)` C++

### explizite vs implizite

explizit: klar definierte Typen, bei Deklaration angegeben  
`(int num = 10)` C++

implizit: Typ wird aus zugewiesenen Wert abgeleitet (`let name = "Max"`)

## 1.2 Das erste Java-Programm

Während Ihrer Berufsschulzeit lernen Sie einige Programmierkonzepte kennen, welche mit der Programmiersprache Java verdeutlicht werden sollen. Dafür sollen Sie nun ein erstes ausführbares Java-Programm erstellen. Ein ausführbares Java-Programm ist eine Klasse mit einer so genannten **main**-Methode. Um eine solche Klasse anzulegen, muss erstmal ein neues Java-Projekt in Eclipse angelegt werden. In dieser main-Methode steht der Programmcode, den das Java-Programm ausführen soll. Soll beispielsweise der Text „Hello World“ auf der Konsole ausgegeben werden, verwendet man beispielsweise den Befehl `System.out.println("Hello World")`:

```
public class MeinErstesProgramm{
    public static void main(String[] args){
        //Im main-Block steht der Programmcode, der ausgeführt werden soll
        System.out.println("Hello World");
        //Weitere Anweisungen...
    }
}
```

**Aufgabe:** Erstellen Sie ein neues Projekt `Einstieg` und legen Sie in diesem Projekt eine Klasse `MeinErstesProgramm` mit einer `main`-Methode an, welche „Hello World“ auf der Konsole ausgibt.

## 1.3 Genauere Betrachtung der Konsolenausgabe

Im folgenden Abschnitt soll die Konsolenausgabe genauer betrachtet werden.

### Aufgaben:

1. Erstellen Sie eine Klasse mit dem Namen `ErweiterteKonsolenausgabe`, die eine `main`-Methode enthält.
2. Führen Sie folgende Zeilen Code in der `main`-Methode aus und beschreiben Sie den Unterschied zwischen den Methoden `System.out.print` und `System.out.println`:

```
System.out.println("Hello");
System.out.println("World");
System.out.print("Hello");
System.out.print("World");
```

3. Führen Sie folgende Zeilen Code in der main-Methode aus und beschreiben Sie die Auswirkung von \n und \t bei folgenden Ausgaben:

```
System.out.print("Hello\n");
System.out.println("World");
System.out.print("Hello\t");
System.out.println("World");
```

\n - next line	Hello
\t - space	World
	Hello World

4. Welche Ausgabe liefern jeweils die dargestellten Programmzeilen:

```
System.out.println("1");
System.out.print("2");
System.out.print("3");
System.out.println("4");
```

1
234

```
System.out.print("1\n");
System.out.print("2");
System.out.print("3\n");
System.out.print("4\t");
```

1
23
4

```
System.out.println("1\n");
System.out.print("2\n");
System.out.print("3\n");
System.out.println("4\n");
```

1
2
3
4

```
System.out.println("1");
System.out.println("2\n");
System.out.print("3");
System.out.println("4");
```

1
2
34

5. Erzeugen Sie mithilfe von System.out.print- bzw. System.out.println-Methoden folgende Konsolenausgabe:

```
\_ _/_ \_ _/
(" ") (" ")
.-----\ / \ /-----.
/ |_____| |_____|\ \
" | \ || || / | " "
```

--

## 1.4 Variablen und Datentypen

In der Programmierung benötigt man sehr oft **Variablen** für Berechnungen, Abläufe,... etc. Jede Variable hat dabei einen Datentyp. Im Folgenden finden Sie einen Überblick über die gängigsten elementaren primitiven Datentypen in Java.

Typ	Bit-Tiefe	Wertebereich
<b>numerisch</b>		
<i>Ganzzahlen</i>		
int	32 Bit	$-2^{31}$ bis $2^{31} - 1$
long	64 Bit	$-2^{63}$ bis $2^{63} - 1$
byte	8 Bit	$-2^7$ bis $2^7 - 1$
(short)	16 Bit	$-2^{15}$ bis $2^{15} - 1$
<i>Fließkommazahlen</i>		
double	64 Bit	$\approx +/- 1,78 \cdot 10^{308}$
(float)	32 Bit	$\approx +/- 3,4 \cdot 10^{38}$
<b>logisch</b>		
boolean	JVM-spezifisch	true, false
<b>Text</b>		
char	16 Bit	Einzelnes Unicode-Zeichen

Neben dem Datentyp benötigt eine Variable einen eindeutigen Namen (keine Systemwörter). Möchte man beispielsweise eine Variable mit dem Namen `alter` vom Datentyp `int`, eine Variable `geschlecht` vom Datentyp `char` oder eine Variable `durchschnitt` vom Datentyp `double` in Java **deklarieren**, geschieht das folgendermaßen:

```
int alter;
char geschlecht;
double durchschnitt;
```

Möchten man diese Variablen nach der Deklaration **initialisieren** bzw. einen Wert zuweisen, so wird dafür in Java der `= Operator` verwendet:

```
int alter;
char geschlecht;
alter = 29;
geschlecht = 'w';
double durchschnitt;
durchschnitt = 2.5;
```

Häufig geschieht die Deklaration und Initialisierung zusammen in einer Zeile:

```
int alter = 29;
char geschlecht = 'w';
double durchschnitt = 2.5;
```

Soll der Inhalt einer Variable auf der Konsole ausgegeben werden, so kann auch hierfür die `System.out.println` bzw. `System.out.print` Methode verwendet werden:

```
int alter = 29;  
System.out.println(alter);
```

Für Zeichenketten bzw. Texte benötigt man den nicht primitiven Datentyp **String**. Strings werden in Anführungszeichen geschrieben:

```
String name = "Robert";
```

### Aufgaben:

1. Erklären Sie kurz, warum folgende Wertzuweisung von der Variable b nicht funktioniert.

```
int x = 24;  
byte b = x;
```

2. Gegeben ist folgender zusammenhängender Codeausschnitt mit Zeilennummern in einer `main`-Methode. Einige Zeilen sind jedoch fehlerhaft. Korrigieren Sie diese.  
Hinweis: Sie müssen insgesamt 9 Zeilen korrigieren

1 int a = 24.5; <input checked="" type="checkbox"/>	9 double f = 32,5; <input checked="" type="checkbox"/>
2 boolean boo = 20; <input checked="" type="checkbox"/>	10 char c = "t"; <input checked="" type="checkbox"/>
3 int g = 17; <input checked="" type="checkbox"/>	11 int m = 2_147_483_647; <input checked="" type="checkbox"/>
4 int y = g; <input checked="" type="checkbox"/>	12 long l = 2_147_483_648; <input checked="" type="checkbox"/>
5 g = g + g; <input checked="" type="checkbox"/>	13 float fl = 32.5; <input checked="" type="checkbox"/>
6 byte b = 3; <input checked="" type="checkbox"/>	14 boolean c = true; <input checked="" type="checkbox"/>
7 byte v = 'b'; <input checked="" type="checkbox"/>	15 String super = "1"; <input checked="" type="checkbox"/>
8 int a = 10; <input checked="" type="checkbox"/>	16 char ch = 65; <input checked="" type="checkbox"/>

3. Was wird bei folgendem Programm auf der Konsole ausgegeben? Erklären Sie die unterschiedliche Bedeutung des + Operators.

```
int i = 5;  
int j = 3;  
String s = "3";  
System.out.println(i + j);  
System.out.println(s + i);  
System.out.println(i + j + s);  
System.out.println(i + s + j);
```

4. Welchen Wert besitzen die Variablen `i` und `j` nach folgenden Zeilen Java-Code? Erklären Sie diese Werte.

```
int i = 2_147_483_647;  
i = i + 1;  
int j = -2_147_483_648;  
j = j - 1;
```

## 1.5 Programmierübung: Notendurchschnitt berechnen

Im Folgenden soll ein Java-Programm entwickelt werden, welches anhand der Noten in einem Fach den entsprechenden Schnitt und damit verbundene Zeugnisnote ermittelt. Dafür wurde schon folgendes Codegerüst erstellt:

```
1 public class Notenberechnung{  
2     public static void main (String[] args){  
3         //Noteninformationen wurden wie folgt eingelesen  
4         int noteTest1 = 3, noteTest2 = 2;  
5         int noteSA1 = 4, noteSA2 = 2;  
6         int anzahl = 6;  
7  
8         int summeTest = noteTest1 + noteTest2;  
9         int summeSA = noteSA1 + noteSA2;  
10  
11         double durchschnitt = (summeTest + summeSA*2)/anzahl;  
12         System.out.println("Ihr Notendurchschnitt beträgt: " + durchschnitt);  
13     }  
14 }
```

$$\frac{17}{6} \approx 2,83$$

1. Obiges Programm liefert keinen korrekten Notendurchschnitt. Erklären Sie kurz warum und korrigieren Sie das Programm. Die Zeilen 4 bis 9 dürfen Sie dabei **nicht** verändern.

2. Die Ausgabe des Notendurchschnitts mit 16 Nachkommastellen ist nicht effektiv. Daher soll der Notendurchschnitt auf zwei Stellen nach dem Komma richtig gerundet werden. Passen Sie Ihr Programm dahingehend an. Der gerundete Durchschnitt soll dabei auch in einer Variable gespeichert werden.

3. Ergänzen Sie Ihr Programm um die Ausgabe der Zeugnisnote, welche sich aus dem Notendurchschnitt ergibt. Beachten Sie dabei, dass bei Komma fünf die bessere Note gegeben werden soll.

Dies bedeutet, dass beispielsweise ein Notendurchschnitt von 2,50 die Zeugnisnote 2 und ein Zeugnisnotendurchschnitt von 2,51 die Zeugnisnote 3 ergeben soll.

**4. Für Schnelle:**

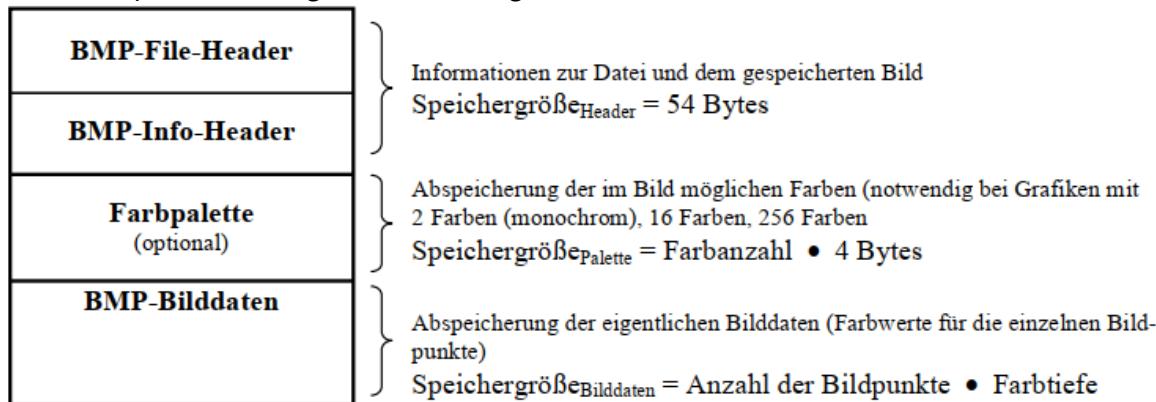
Ergänzen Sie Ihr Programm um die Ausgabe des entsprechenden Zeugnistext für die jeweilige Zeugnisnote:

Zeugnisnote	Zeugnistext
1	„sehr gut“
2	„gut“
3	„befriedigend“
4	„ausreichend“
5	„mangelhaft“
6	„ungenügend“

## BitmapRechner

Eine Bitmap ist ein unkomprimiertes Standardformat für Pixelgrafiken im Windows-Umfeld. bmp-Dateien besitzen dabei einen einheitlichen Aufbau, der notwendig ist, damit die Datei nicht nur vom abspeichernden Programm interpretiert und angezeigt werden kann.

Eine Bitmap-Datei besitzt grundsätzlich folgenden Aufbau:



Abspeicherung der im Bild möglichen Farben (notwendig bei Grafiken mit 2 Farben (monochrom), 16 Farben, 256 Farben)  
 $\text{Speichergröße}_{\text{Palette}} = \text{Farbanzahl} \cdot 4 \text{ Bytes}$

Abspeicherung der eigentlichen Bilddaten (Farbwerte für die einzelnen Bildpunkte)  
 $\text{Speichergröße}_{\text{Bilddaten}} = \text{Anzahl der Bildpunkte} \cdot \text{Farbtiefe}$

Anzahl der Bildpunkte:  
 $\text{horizontale Bildpunkte} \cdot \text{vertikale Bildpunkte}$

Farbtiefe:  
 Die Farbtiefe gibt den Speicherbedarf (in Bit) je Bildpunkt an. Dabei gibt sie die Anzahl der möglichen Farben wieder.

Anzahl der Farben	Speicherbedarf je Bildpunkt	
monochrom	1 Bit	s/w-Grafik
16 Farben	4 Bit	Standard-VGA-Farbtiefe
256 Farben	8 Bit	„High-Color“
65536 Farben	16 Bit	„True-Color“
16,7 Mio Farben	24 Bit	

## Arbeitsauftrag

- Erstellen Sie ein ausführbares Java-Programm, das aufgrund der Eigenschaften Breite, Länge und Farbtiefe die entsprechende Dateigröße der bmp-Datei in KiB berechnen kann.  
 Verwenden Sie zum Abspeichern der Variablen den Datentyp **int**!
- Erweitern Sie Ihr Programm, indem Sie die Möglichkeit zur Eingabe der Variablen und die Einheit der Dateigröße über die Konsole ermöglichen.



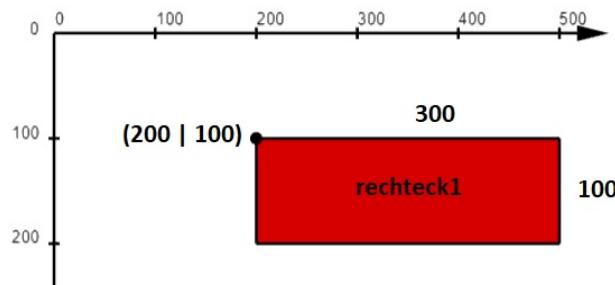
**Hinweis:** Informieren Sie sich über die Klasse **Scanner** im Internet.

## 2 Theoretische Grundlagen der Objektorientierung

### 2.1 Objekte

Wie der Name schon sagt, sind so genannte Objekte bei der Objektorientierung ein zentraler Begriff. Objekte können allgemein Dinge (z.B. Räume, Autos,...), Lebewesen (z.B. Menschen, Tiere,...) oder Begriffe (z.B. Konten, Buchungen,...). Jedes Objekt kann durch Eigenschaften beschrieben werden, diese Eigenschaften eines Objekts werden **Attribute** genannt. Zum Beispiel sind dies bei einem Rechteck läng, breite, füllfarbe,... (Hinweis: Attribute werden in der Programmierung klein geschrieben)

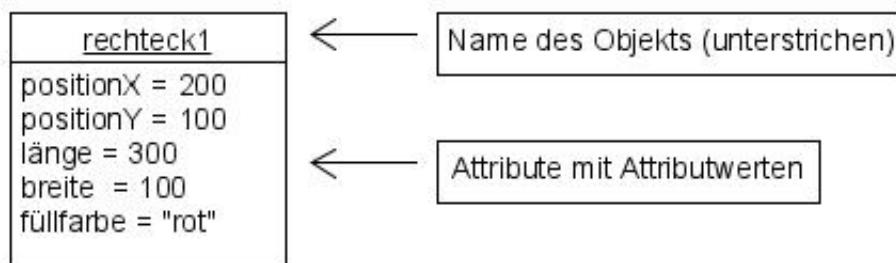
Ein konkreter Wert für ein Attribut wird **Attributwert** genannt. Betrachtet man folgendes beispielhafte Objekt `rechteck1`:



So könnte dieses Objekt durch folgende Attribute beschrieben werden:

`positionX`, `positionY` (für die linke obere Ecke), `länge`, `breite` und `füllfarbe`. Die Attributwerte für das Objekt `rechteck1` wären dann für `positionX` 100, für `positionY` 200, für `länge` 300, für `breite` 100 und für `füllfarbe` „rot“.

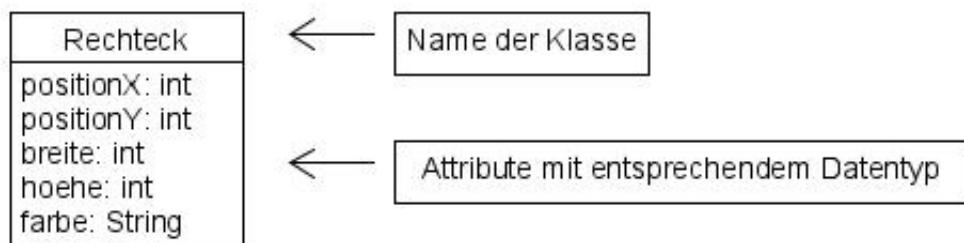
Um diese Informationen übersichtlich darzustellen, verwendet man in UML **Objektkarten** bzw. **Objektdiagramme**:



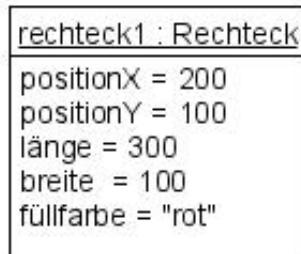
Hinweis: Sollten Sie Objektkarten per Hand zeichnen, achten Sie darauf, dass Sie den Objektnamen unterstreichen und nicht durchstreichen.

## 2.2 Klassen

Klassen sind Bau- bzw. Konstruktionspläne für gleichartige Objekte. Sie dienen dabei als Vorlage zur Erzeugung von neuen Objekten. Eine Klasse besitzt unter anderem eine Liste von Attributen mit deren dazugehörigem Datentyp. Jedes Objekt derselben Klasse besitzt die gleichen Attribute, die Attributwerte der Objekte können sich aber unterscheiden. Die Informationen einer Klasse werden übersichtlich in einer **Klassenkarte** dargestellt. Beispielsweise wäre ein Bauplan für das Objekt `rechteck1` die Klasse `Rechteck` mit folgender Klassenkarte:



Dabei werden die Objektkarten um den dazugehörigen Klassennamen wie folgt erweitert:



Wenn Sie bei den nachfolgenden Aufgaben Objektkarten bzw. Klassenkarten zeichnen müssen, verwenden Sie das Werkzeug UMLet.

### Aufgaben:

1. (a) Erklären Sie kurz in eigenen Worten den Unterschied zwischen Attribut und Attributwert.

*Datentyp Attribut = Attributwert*

- (b) Erklären Sie kurz in eigenen Worten den Zusammenhang zwischen einem Objekt und einer Klasse.

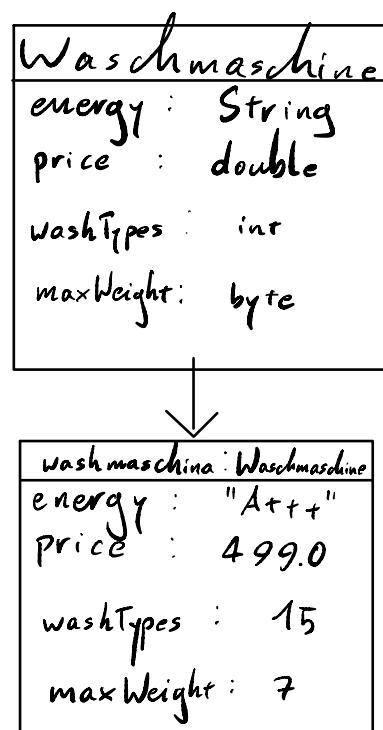
*Klasse = Mutter  
Objekt = Kind*

2. Gegeben ist folgender Ausschnitt eines Angebots für eine Waschmaschine.



Abbildung 1: Schulbuch Informatik 1A (Klett Verlag), S.14

- (a) Zeichnen Sie eine Klassenkarte für die angebotene Waschmaschine mit vier Attributen. Dabei sollen alle Attribute einen unterschiedlichen Datentyp besitzen.
- (b) Zeichnen Sie mit der erstellten Klassenkarte eine entsprechende Objektkarte für die angebotene Waschmaschine.



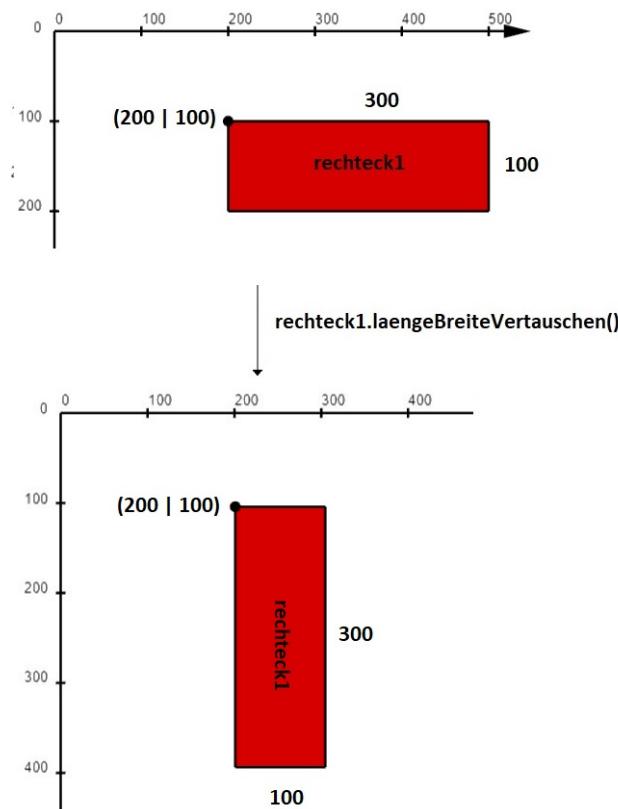
## 2.3 Methoden

Methoden sind „Tätigkeiten“, die Objekte ausführen können. Diese Tätigkeiten werden in der Klasse des Objekts definiert. Im Folgenden soll die Klasse `Rechteck` um verschiedene Methoden erweitert werden.

### 2.3.1 Länge und Breite vertauschen

Möchte man bei Rechtecken die Länge und Breite vertauschen, so muss in der Klasse `Rechteck` dafür eine Methode `laengeBreiteVertauschen` (Methodennamen sollten in der Regel wie Attributnamen mit einem Kleinbuchstaben beginnen) definiert werden. Soll dann anschließend das Objekt `rechteck1` diese Methode ausführen, so erfolgt die Anweisung über den **Punktoperator**:

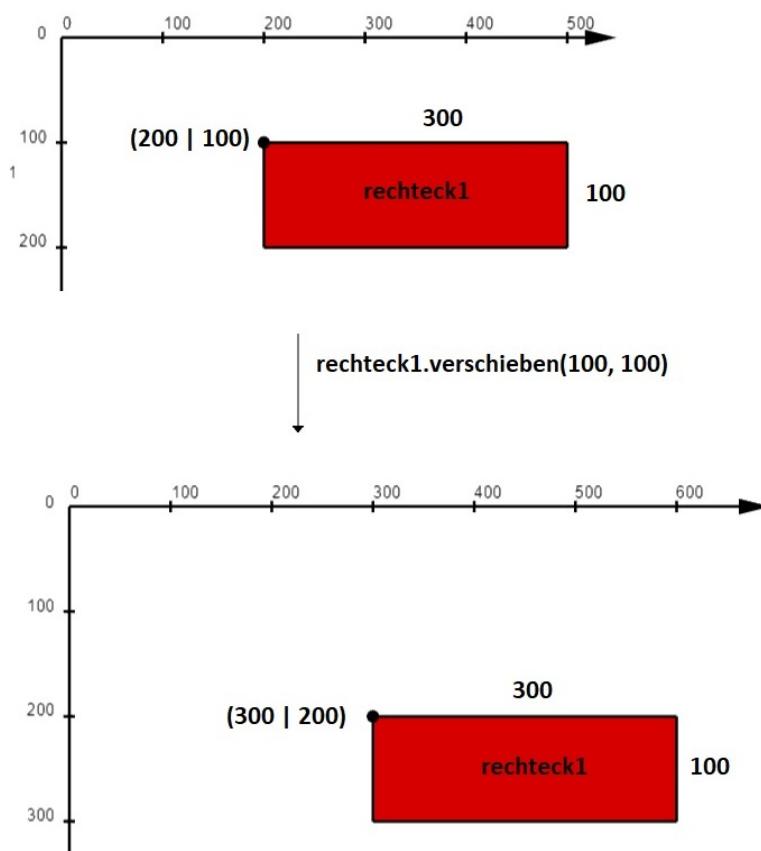
`rechteck1.laengeBreiteVertauschen()`



### 2.3.2 Verschieben

Sollen nun zusätzlich Rechteck-Objekte verschoben werden können, so muss die Klasse Rechteck eine Methode **verschieben** bereit stellen. Der Unterschied zum ersten Beispiel ist, dass hierfür noch weitere Informationen übergeben werden müssen: Um wie viel das Rechteck in x- bzw. y-Richtung verschoben werden soll. Diese übergebenen Werte werden auch **Übergabeparameter** genannt und stehen beim Methodenaufruf in den runden Klammern. Mehrere Parameter werden durch Kommas getrennt bzw. bleiben die Klammern leer, falls es keine Übergabeparameter gibt (siehe Beispiel 1). Ein Aufruf könnte dann beispielsweise wie folgt aussehen:

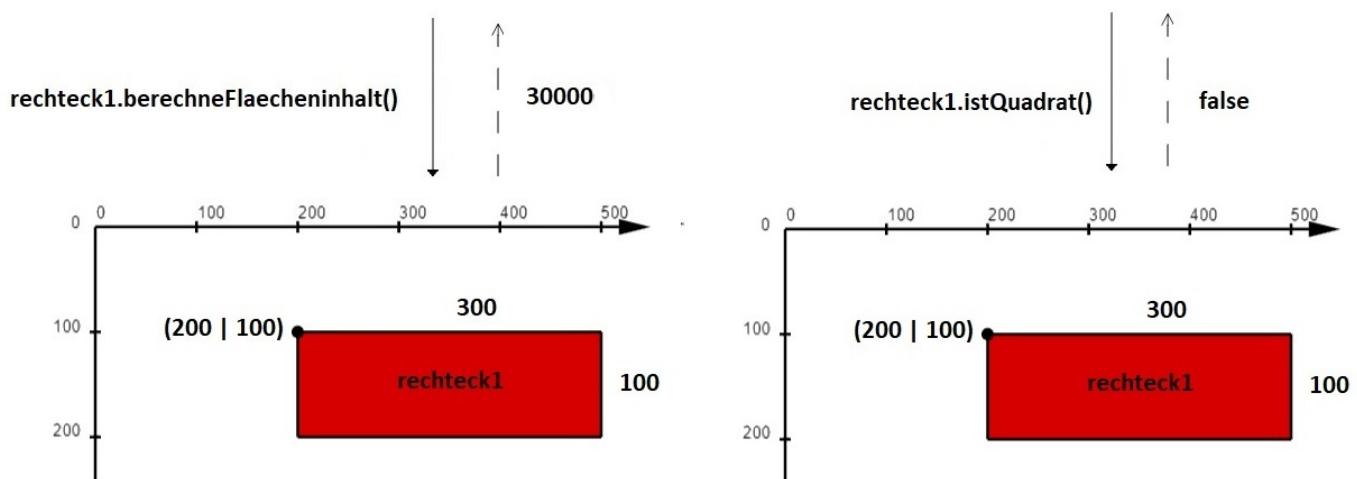
```
rechteck1.verschieben(100, 100)
```



### 2.3.3 Informationen über Objekte

Häufig benötigt man in der Programmierung Informationen über Objekte. Bei Rechtecken könnte dies beispielsweise die Größe des Flächeninhalts sein oder ob es sich bei dem Rechteck um ein Quadrat handelt. Um an diese Informationen zu kommen, werden häufig Methoden verwendet.

Dies bedeutet, dass diese Methode einen Wert für eine Weiterverarbeitung zurückgeben müssen. Das heißt, eine Methode für die Größe des Flächeninhalts würde ein Wert vom Datentyp `int` (**Rückgabedatentyp**) zurückgeben, eine Methode, ob ein Rechteck ein Quadrat ist, ein Wert vom Datentyp `boolean`, welcher für die Weiterverarbeitung gespeichert werden kann:



Codebeispiel:

```
int flaecheninhalt = rechteck1.berechneFlaecheninhalt()
boolean istQuadrat = rechteck1.istQuadrat()
```

Bei Methoden muss immer ein Rückgabedatentyp angegeben werden. Gibt eine Methode keinen Wert zurück, so verwendet man als Rückgabedatentyp das Schlüsselwort **void**. Diesen Rückgabedatentyp `void` würde man bei den beiden vorherigen Methoden `laengeBreiteVertauschen` und `verschieben` verwenden.

Die Information über die Methoden einer Klasse werden nach folgendem Muster in die Klassenkarte mit aufgenommen:

### Methodename(Übergabeparameter mit Datentyp): Rückgabedatentyp

Den Teil Methodename(Übergabeparameter mit Datentyp) nennt man dabei **Methode signatur**.

Die bisherige Klassenkarte der Klasse Rechteck wird damit um die neu definierten Methoden wie folgt erweitert:

Rechteck
positionX: int positionY: int breite: int hoehe: int farbe: String
laengeBreiteVertauschen(): void
verschieben(xRichtung:int, yRichtung:int): void
istQuadrat(): boolean
berechneFlächeninhalt(): int

### Aufgaben:

1. Nennen Sie einen Vorteil, wenn für die Größe des Flächeninhalts kein Attribut angelegt wird, sondern eine Methode mit Rückgabedatentyp verwendet wird. Unter welchen Umständen wäre ein Attribut für die Größe des Flächeninhalts besser geeignet als eine Methode?

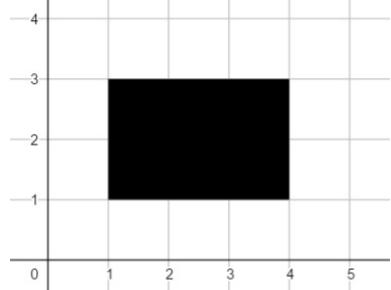
Es braucht keine Parameter, weil er diese als Attribute hat.

Abspeichern > wiederholtes aufrufen

2. Im Folgenden sehen Sie ein Objekt *r1* zu vier verschiedenen Zeitpunkten (Zeitpunkt 1 bis Zeitpunkt 4). Dieses Objekt besitzt dabei folgende Methodensignaturen:

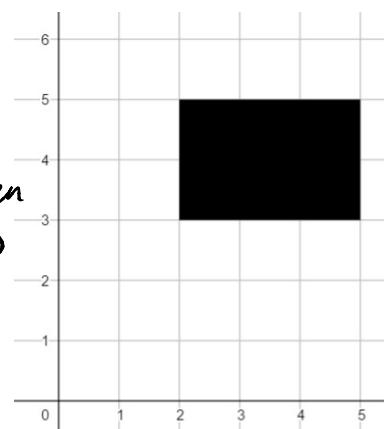
- *setFüllfarbe(farbe:String)*
- *setBreite(breite:String)*
- *setLinienart(linienart:String)*
- *drehen(drehwinkel:int)*
- *verschieben(xRichtung:int, yRichtung:int)*

Geben Sie jeweils einen Methodenaufruf in der Punktnotation an, der zwischen den abgebildeten Zeitpunkten geschehen ist:

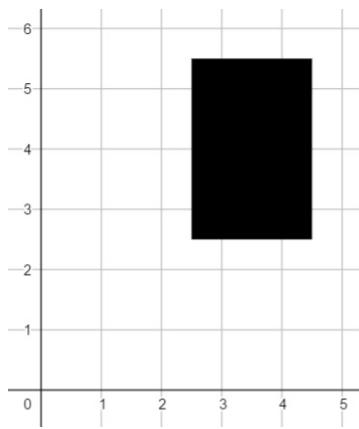


Zeitpunkt 1

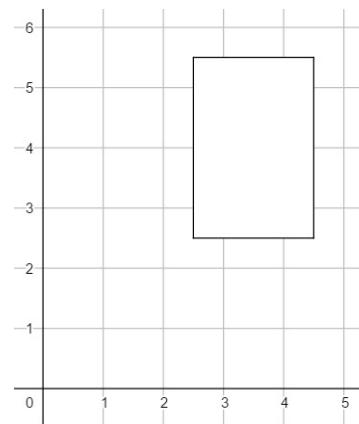
*verschieben*  
→



Zeitpunkt 2



Zeitpunkt 3



Zeitpunkt 4

Methodenaufruf von Zeitpunkt 1 nach Zeitpunkt 2:

*r1.verschieben(1, 2);*

Methodenaufruf von Zeitpunkt 2 nach Zeitpunkt 3:

*r1.drehen(90);*

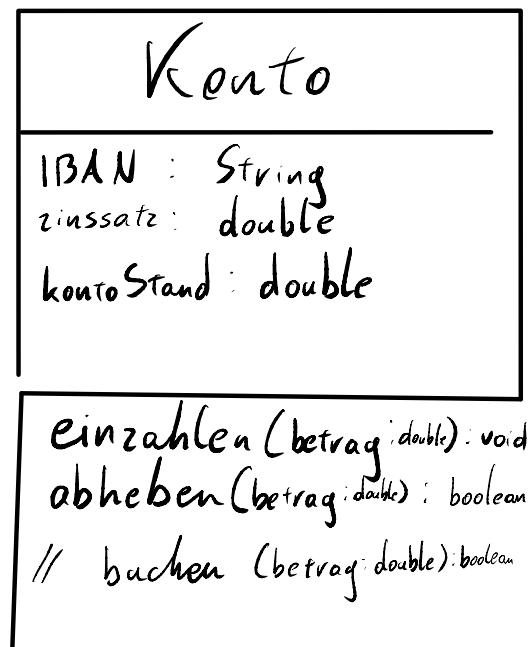
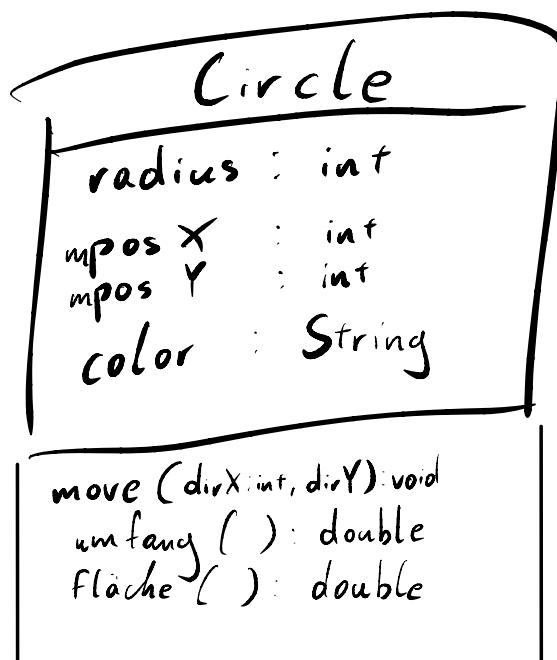
Methodenaufruf von Zeitpunkt 3 nach Zeitpunkt 4:

*r1.setFüllfarbe("weiß");*

3. Erstellen Sie für folgende Situationen jeweils eine UML-Klassenkarte mit Attributen und Methoden.

(a) Ein Kreis besitzt einen **ganzzahligen Radius** und einen **Mittelpunkt** mit ebenfalls **ganzzahligen Koordinaten**. Außerdem besitzt er eine **Füllfarbe**. Kreise sollen **verschoben** werden können, sowie **Information über den Umfang und Flächeninhalt** bereitstellen.

(b) Ein Konto besitzt einen Kontostand, **einen Zinssatz (in Prozent)**, sowie eine **IBAN**. Bei einem Konto sollen **Beträge eingezahlt** werden können, sowie **Beträge abgehoben** werden können. Das **Abheben** soll aber nur passieren, wenn der **Kontostand größer gleich als der zu abhebende Betrag ist**. Ob der **Abhebevorgang** dementsprechend erfolgreich war, soll durch eine entsprechende Rückgabe **signalisiert** werden.



### 3 Umsetzung in Java (ohne Methoden)

#### 3.1 Von der Klassenkarte zur Java-Klasse

Die Klassenkarte eines Rechtecks soll nun in Java umgesetzt werden. Die Methoden werden dabei zu einem späteren Zeitpunkt umgesetzt.

Rechteck
positionX: int
positionY: int
breite: int
hoehe: int
farbe: String

Die Attribute werden mit ihrem Datentyp dabei zeilenweise wie folgt für eine Java-Klasse übersetzt:

```
public class Rechteck{  
    int positionX;  
    int positionY;  
    int breite;  
    int hoehe;  
    String farbe;  
}
```

Attribute mit gleichem Datentyp können dabei in einer Zeile zusammengefasst werden:

```
public class Rechteck{  
    int positionX, postionY, breite, hoehe;  
    String farbe;  
}
```

Aus Gründen der Übersichtlichkeit ist davon aber eher abzuraten.

**Hinweis:** Achten Sie bei der Umsetzung in den folgenden Aufgaben darauf, dass sie Attribute exakt so schreiben, wie sie in der Klassenkarte stehen.

**Aufgaben:**

1. Erstellen Sie ein neues Java-Projekt `Einstieg_OOP` und dort im `src`-Ordner ein neues Package `figuren`. Achten Sie darauf, dass Sie mindestens Java-Version 17 für das Projekt eingestellt haben.
2. Erstellen Sie in diesem Package eine neue Klasse `Rechteck` mit den obigen Attributnen.
3. Erstellen Sie in diesem Package ebenfalls eine weitere Java-Klasse für folgende Klassenskarte (ohne Methoden).

Kreis
positionX: int
positionY: int
radius: int
farbe: String

## 3.2 Erzeugen von Objekten: Der Konstruktor und Punktoperator

### 3.2.1 Erste Schritte

Um nun Figuren zeichnen zu lassen, müssen Objekte der jeweiligen Figur-Klasse erzeugt werden.

Ein neues Objekt der Klasse Rechteck `rechteck1` kann mit dem Schlüsselwort `new` wie folgt erzeugt werden:

```
Rechteck rechteck1 = new Rechteck();
```

Mit `new Rechteck()` wird dabei der so genannte **Konstruktor** der Klasse Rechteck aufgerufen, welcher ein neues Objekt der Klasse Rechteck im Arbeitsspeicher „erzeugt“. Der Konstruktor ist dabei eine Art Vorschrift, wie ein Objekt dieser Klasse erzeugt werden soll.

Auf die Attributwerte des Objekts `rechteck1` kann mit dem **Punktoperator** folgendermaßen zugegriffen werden.

```
rechteck1.breite  
rechteck1.farbe  
...
```

Allgemein: **Objektname.Attributname**

#### Aufgaben:

1. Erstellen Sie in Ihrem Package eine neue Klasse Test mit einer main-Methode.
2. Erzeugen Sie in dieser main-Methode ein neues Objekt `rechteck1` der Klasse Rechteck.
3. Geben Sie nun alle Attributwerte des Objekts `rechteck1` auf der Konsole aus. Was fällt Ihnen auf?

*NULL, 0, null*

### 3.2.2 Genauere Betrachtung des Konstruktors

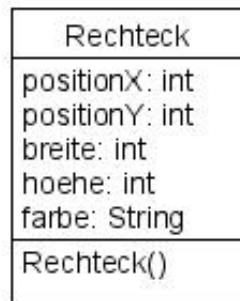
Wird für eine Klasse kein expliziter Konstruktor angegeben, so werden für die Attribute bei der Objekterzeugung Standardwerte verwendet (0 für int, false für boolean, null für String,...). Um nun für jedes Attribut sinnvolle Attributwerte bei der Objekterzeugung festzulegen, muss in der Klasse Rechteck explizit ein Konstruktor angegeben werden. Dieser könnte beispielsweise folgendermaßen aussehen:

```
Rechteck() {
    positionX = 200;
    positionY = 150;
    farbe = "rot";
    //... Weitere Zuweisungen
}
```

Wird nun ein Objekt der Klasse Rechteck erstellt, so besitzt das Objekt von Anfang an den Attributwert 200 für das Attribut positionX, den Wert „rot“ für das Attribut farbe,..., wichtig ist dabei:

**Konstruktoren müssen exakt so geschrieben werden wie der Klassenname.**

Konstruktoren werden als Methode ohne Rückgabetyp in Klassenkarten gekennzeichnet:



#### Aufgaben:

- Was ist die Aufgabe eines explizit angegebenen Konstruktors?  
*Das setzen der Wert zu den jeweiligen Attribu*
- Ergänzen Sie die Klasse Rechteck um einen Konstruktor, so dass bei der Objekterzeugung die linke obere Ecke bei (200|150) liegt und das Objekt die Höhe 100, die Breite 300 und die Farbe rot besitzt.

3. Damit Ihre Implementierung grafisch getestet werden kann, müssen sie die jar-Datei `shapes.jar` aus dem Klassenlaufwerk in Ihr Projekt einbinden. Gehen Sie dabei in Eclipse wie folgt vor:

- (a) Erstellen Sie mit einem Rechtsklick auf Ihren Projektordner einen neuen Ordner (`New → Folder`) mit dem Namen `lib`.
- (b) Kopieren Sie die Datei `shapes.jar` aus dem Klassenlaufwerk in diesen Ordner. (Hinweis: Eclipse unterstützt dies per Drag and Drop).
- (c) Nun muss diese Datei dem Classpath hinzugefügt werden. Dies funktioniert folgendermaßen:  
Rechtsklick in Eclipse auf die Datei `shapes.jar` → `Build Path → Add to Build Path` auswählen.

Hinweis: Mögliche Anleitung für IntelliJ und VS-Code.

4. Erstellen Sie in der `main`-Methode Ihrer Klasse `Test` aus dem vorherigen Aufgabenblock, analog zum Objekt `rechteck1` ein neues Objekt `leinwand` der Klasse `Leinwand`.

Hinweis: Damit Sie Objekte der Klasse `Leinwand` in einer Klasse verwenden können, müssen Sie folgende Import-Anweisung vor der Klassendefinition Ihrer Klassen einfügen:

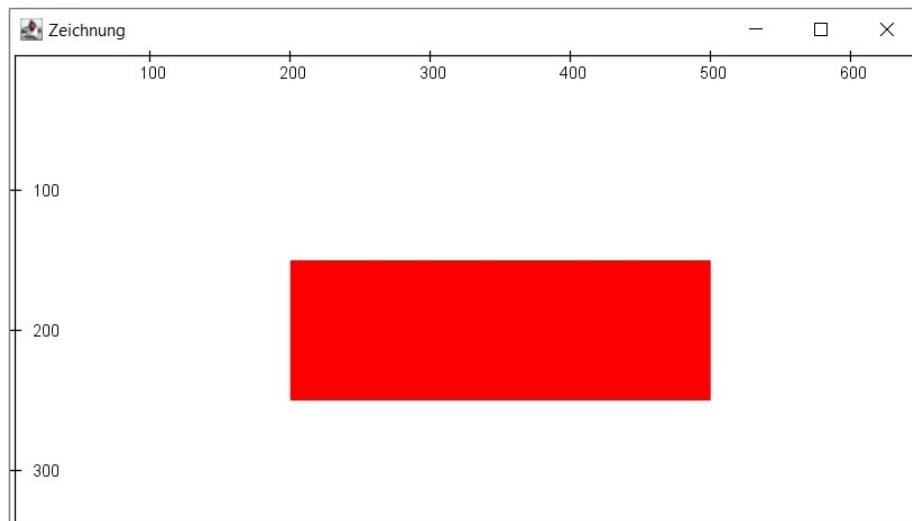
```
import ack.shapes.Leinwand;

public class Test{
    ...
}
```

5. Testen Sie Ihren Konstruktor der Klasse `Rechteck`, indem Sie Ihr Objekt `rechteck1` mit der Methode `zeichne` des Objekts `leinwand` zeichnen lassen:

```
leinwand.zeichne(rechteck1);
```

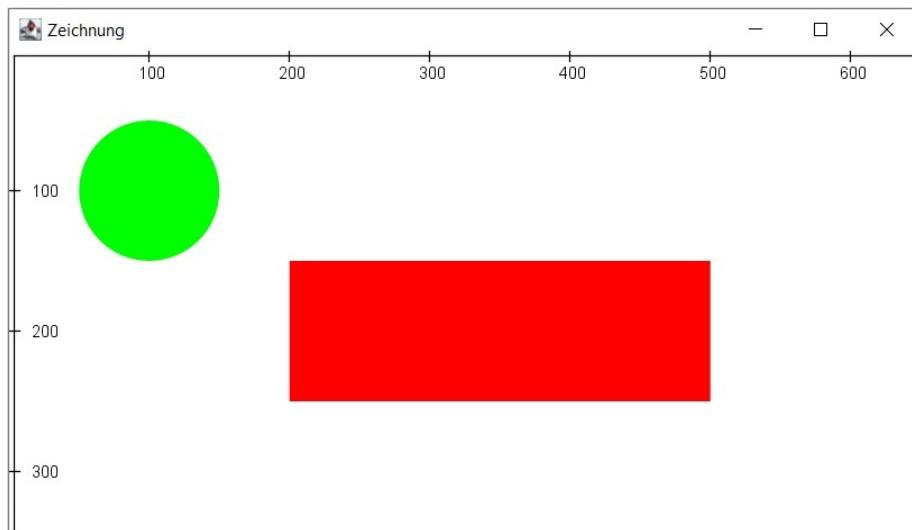
Wenn Sie nun das Programm ausführen, sollte dabei das Rechteck folgendermaßen gezeichnet werden:



6. Erstellen Sie nun analog einen Konstruktor für die Klasse `Kreis`, so dass bei der Objekterzeugung der Mittelpunkt bei (100|100) liegt und das Objekt den Radius 50 und die Farbe „gruen“ besitzt.

7. Testen Sie Ihren Konstruktor, indem Sie in der main-Methode Ihrer Klasse Test ein weiteres Objekt kreis1 der Klasse Kreis erzeugen und es ebenfalls mit der Methode zeichne des Objekts leinwand zeichnen lassen.

Dabei sollte nach der Ausführung die Zeichnung wie folgt aussehen:

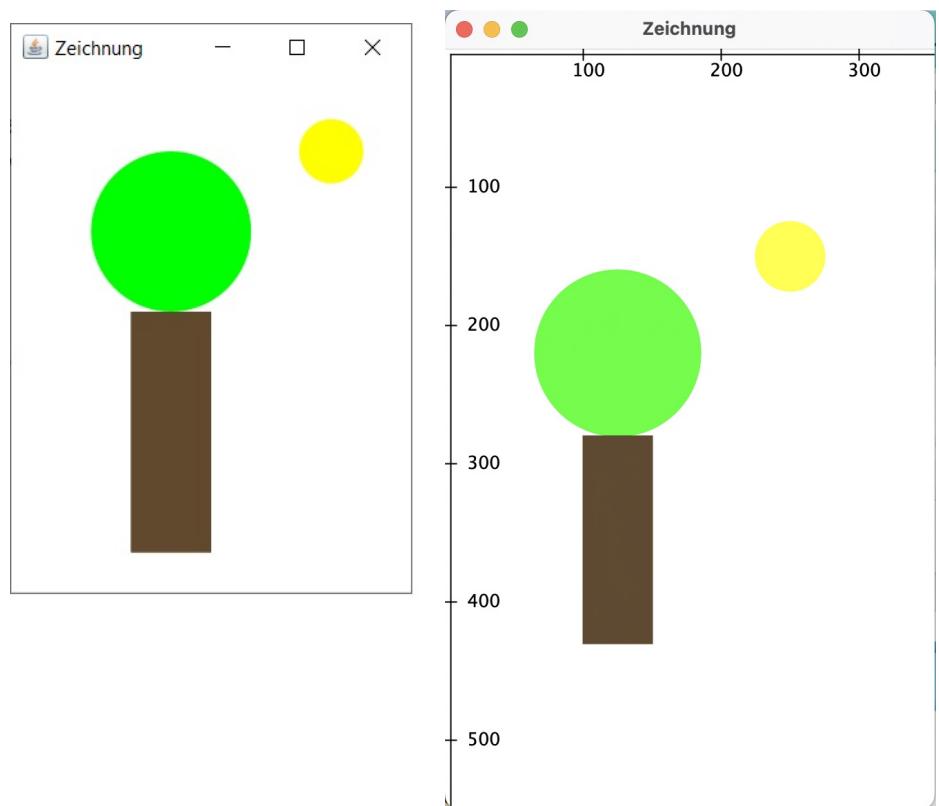


### 3.2.3 Erste Zeichnung

Nun können Sie eine erste richtige Zeichnung anfertigen.

#### Aufgaben:

1. Erstellen Sie eine neue Klasse `ErsteZeichnung` mit einer `main`-Methode.
2. Legen Sie in der `main`-Methode geeignete Objekte der Klassen `Leinwand`, `Rechteck` und `Kreis` an, so dass folgendes Bild gezeichnet wird. Überlegen Sie sich mit einer Skizze passende Attributwerte für Ihre Objekte und verwenden Sie den Punktoperator, um die passenden Attributwerte zuzuweisen.



### 3.2.4 Mehrere Konstruktoren für eine Klasse

Beim Zeichnen ist das passende Setzen jedes einzelnen Attributwertes für jedes neue Objekt ziemlich umständlich. Einfacher ist es, bei der Objekterzeugung die passenden Werte direkt dem Konstruktor als Parameter zu übergeben. Anstelle den bisherigen Konstruktor zu ändern, kann man dafür einen weiteren Konstruktor definieren:

```
Rechteck(int positionX, int positionY, int breite, int hoehe,  
         String farbe){  
    this.positionX = positionX;  
    this.positionY = positionY;  
    //Weitere Zuweisungen...  
}
```

Hierbei ist zu beachten, dass das Schlüsselwort `this` notwendig ist. Mit diesem Schlüsselwort stellt man eine Referenz zur aktuellen Instanz her, d.h. mit `this.positionX` stellt man einen Bezug zum Attribut `positionX` der Klasse her. Dies muss in diesem Fall gemacht werden, um den Namenskonflikt aufzuheben bzw. da der übergebene Parameter das Klassenattribut verdeckt.

Besitzt eine Klasse mehrere Konstruktoren, nennt man dies **überladen von Konstruktoren**. Wichtig ist dabei, dass sich alle Konstruktoren einer Klasse in den Übergabeparametern unterscheiden.

Dieser Konstruktor wird ebenfalls in die Klassenkarte mit aufgenommen:

Rechteck
positionX: int positionY: int breite: int hoehe: int farbe: String
Rechteck() Rechteck(positionX:int, positionY:int, breite:int, hoehe:int, farbe:String)

**Aufgaben:**

1. Die Klasse Leinwand besitzt neben dem Konstruktor ohne Parameter einen weiteren Konstruktor. Beschreiben Sie kurz den weiteren Konstruktor.

Der Unterschied zwischen Leinwand mit und ohne Parameter ist, dass der ohne Parameter den Konstruktor `it` Parameter aufruft mit `this()`; Dieser wird mit vorgeschriebenen Paramater erstellt

2. Ergänzen Sie die Klasse Rechteck um einen weiteren Konstruktor `Rechteck (int positionX, int positionY, int breite, int hoehe, String farbe)`, welcher die Attributwerte auf die entsprechenden übergebenen Werte setzt. Ergänzen Sie die Klasse Kreis um einen analogen Konstruktor.
3. Benutzen Sie nun die neuen Konstruktoren, um den Programmcode aus dem vorherigen Aufgabenblock in der Klasse ErsteZeichnung zu vereinfachen.
4. Vereinfachen Sie in den Klassen `Rechteck` und `Kreis` die Konstruktoren ohne Parameter, indem diese Konstruktoren den jeweilige Konstruktor mit Parametern aufrufen. Dies ist ebenfalls mit dem Schlüsselwort `this` möglich.

## 4 Implementierung von Methoden

Nachdem Sie sich nun ausführlich mit Attributen beschäftigt haben, geht es nun um die Implementierung von Methoden. Die Klassenkarte der Klasse Rechteck wurde dafür um die zu implementierenden Methoden erweitert:

Rechteck
positionX: int
positionY: int
breite: int
hoehe: int
farbe: String
Rechteck()
Rechteck(positionX:int, positionY:int, breite:int, hoehe:int, farbe:String)
verschieben(xRichtung:int, yRichtung:int): void
flaecheninhaltBerechnen(): double
hoeheBreiteVertauschen(): void
vergroessern(faktor:int): void
istQuadrat(): boolean
umfangBerechnen(): double

Eine Methode wird dabei nach folgendem Muster implementiert:

```
Rückgabedatentyp Methodename (Übergabeparameter mit Datentyp) //Methodenkopf
{
    //Funktionalität der Methode: Methodenrumpf
}
```

Eine Implementierung der Methode verschieben könnte dabei wie folgt aussehen:

*public void verschieben (int xRichtung, int yRichtung) {  
 x += xRichtung;  
 y += yRichtung;  
 }*

Für Methoden, die einen Wert zurückgeben, muss das Schlüsselwort **return** verwendet werden. Dies wäre beispielsweise bei der Implementierung der Methode

flaecheninhaltBerechnen notwendig:

*public double flaecheBerechnen() {  
 double flaeche = hoehe \* breite;  
 return flaeche;  
 }*

Wichtig ist, dass die **return**-Zeile die letzte Anweisung in der Methode ist.

**Aufgaben:**

1. Implementieren Sie alle restlichen Methoden aus der obigen Klassenkarte in Ihrer bisherigen Klasse Rechteck. Die Methode vergroessern soll dabei das Rechteck um den übergebenen Faktor vergrößern.

Hinweis: Mit dem Operator == können in Java zwei Werte eines primitiven Datentyps verglichen werden.

- vergroessern(faktor: int): void  
`public void vergroessern (int faktor) {  
 hoehe *= faktor;  
 breite *= faktor; }`
- umfangBerechnen(): double  
`public double umfang berechnen() {  
 return 2 * hoehe + 2 * breite;  
}`
- istQuadrat(): boolean  
`public boolean ist Quadrat () {  
 return hoehe = breite;  
}`
- hoeheBreiteVertauschen(): void  
`public void hoehe Breite Vertauschen () {  
 int temp = hoehe;  
 hoehe = breite;  
 breite = temp;`

2. Erstellen sie zum Testen Ihrer Implementierung eine neue Klasse MethodenTest mit einer main-Methode. Erstellen Sie dort ein neues blaues Rechteck rechteck1 an der Position (100|200) mit einer Länge von 100 und einer Breite von 50.

Überprüfen Sie mithilfe von Konsolenausgaben, ob die Methoden istQuadrat, flaecheninhaltBerechnen und umfangBerechnen beim Objekt rechteck1 folgende Werte zurückgeben:

```
false
5000.0
300.0
```

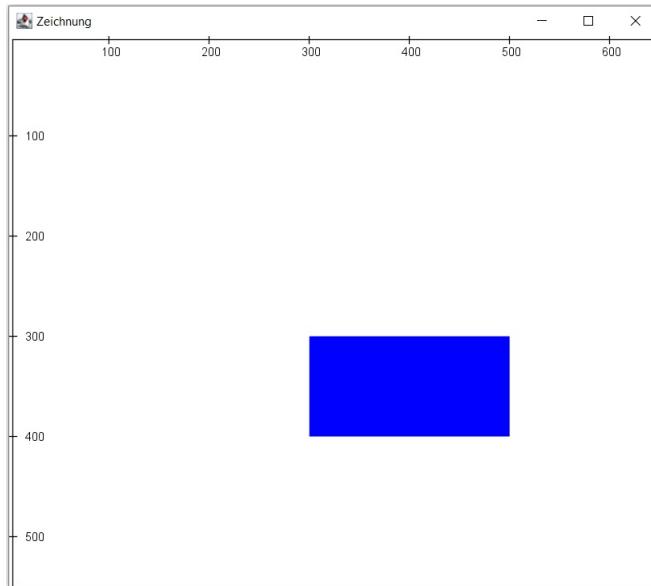
Erstellen Sie anschließend in dieser Klasse ein neues Objekt leinwand der Klasse Leinwand. Die restlichen Methoden, die die Attributwerte eines Objekts verändern, können Sie nach folgendem Muster testen:

```
leinwand.zeichne(rechteck1); // Zeichnen des Rechtecks vor Methodenaufruf
rechteck1.verschieben(100,100); // Ausführen der Methode
leinwand.warte(2000); // Warte 2 Sekunden
leinwand.zeichne(rechteck1); // Zeichnen des Rechtecks nach der Veränderung

rechteck1.vergroessern(2);
leinwand.warte(2000);
leinwand.zeichne(rechteck1);

rechteck1.hoeheBreiteVertauschen();
leinwand.warte(2000);
leinwand.zeichne(rechteck1);
```

Danach sollte Ihr Rechteck auf der Leinwand wie folgt aussehen:



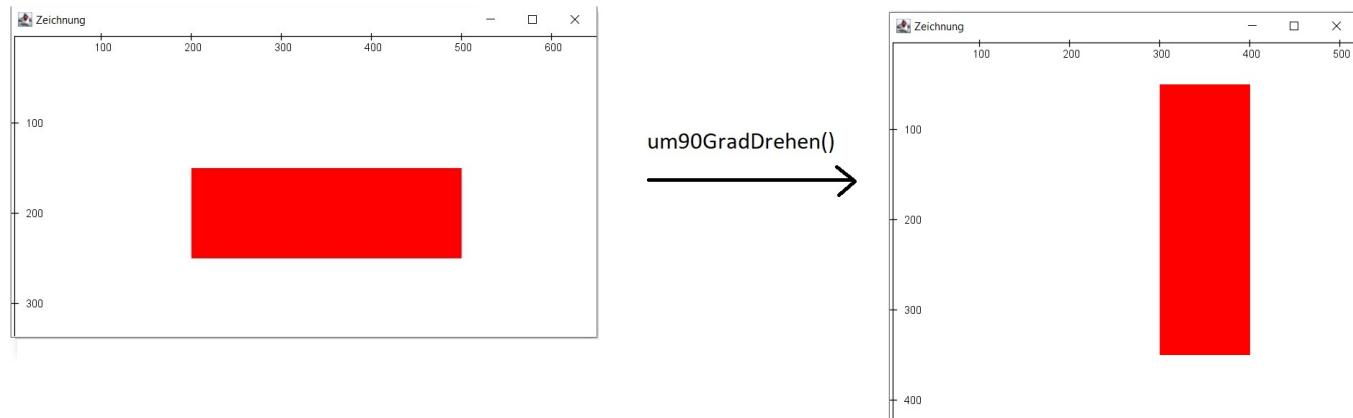
3. Implementieren Sie in der Klasse Kreis ebenfalls folgende Methoden:

- `verschieben(xRichtung: int, yRichtung: int): void`
- `vergroessern(zusaetzlicherRadius: int): void`
- `flaecheninhaltBerechnen(): double`
- `umfangBerechnen(): double`

Testen Sie Ihre Implementierung selbstständig analog zur vorherigen Aufgabe. Bei einem Kreis mit einem Radius von 100 sollte für den Umfang ca. 628,32 und für den Flächeninhalt ca. 31415,93 herauskommen.

#### 4. Für Experten

- (a) Implementieren Sie eine Methode `void um90GradDrehen()`, welche das Rechteck um 90 Grad um den Mittelpunkt des Rechtecks dreht:



Testen Sie anschließend Ihre Implementierung, indem Sie ein Rechteck vier-mal um 90 Grad drehen lassen und überprüfen, ob sich das Rechteck wieder im Ausgangszustand befindet.

- (b) Animationen

- Erstellen Sie eine neue Klasse `EinfacheAnimation` mit einer `main`-Methode und versuchen Sie eine einfache Animation nach dem unten beschriebenen Verfahren zu erzeugen. Verändern Sie anschließend mal die Werte für die Verschiebung oder den Parameter der Methode `warten` und beobachten Sie, wie sich die Animation dabei verändert:

Mithilfe einer Endlosschleife lassen sich nun auch einfache Animationen umsetzen. Hierfür muss beispielsweise ein Kreis mit einem entsprechenden Objekt der Klasse `Leinwand` gezeichnet werden, anschließend der Kreis minimal verschoben werden (beispielsweise 5 Einheiten in x- bzw. y-Richtung) und bevor der Kreis erneut gezeichnet wird, die Methode `warte` mit einem Wert von beispielsweise 30 auf dem entsprechenden Objekt der Klasse `Leinwand` aufgerufen wird. Durch die Wiederholung dieser Anweisungen entsteht eine einfache Animation.

**ii. Anspruchsvoll:**

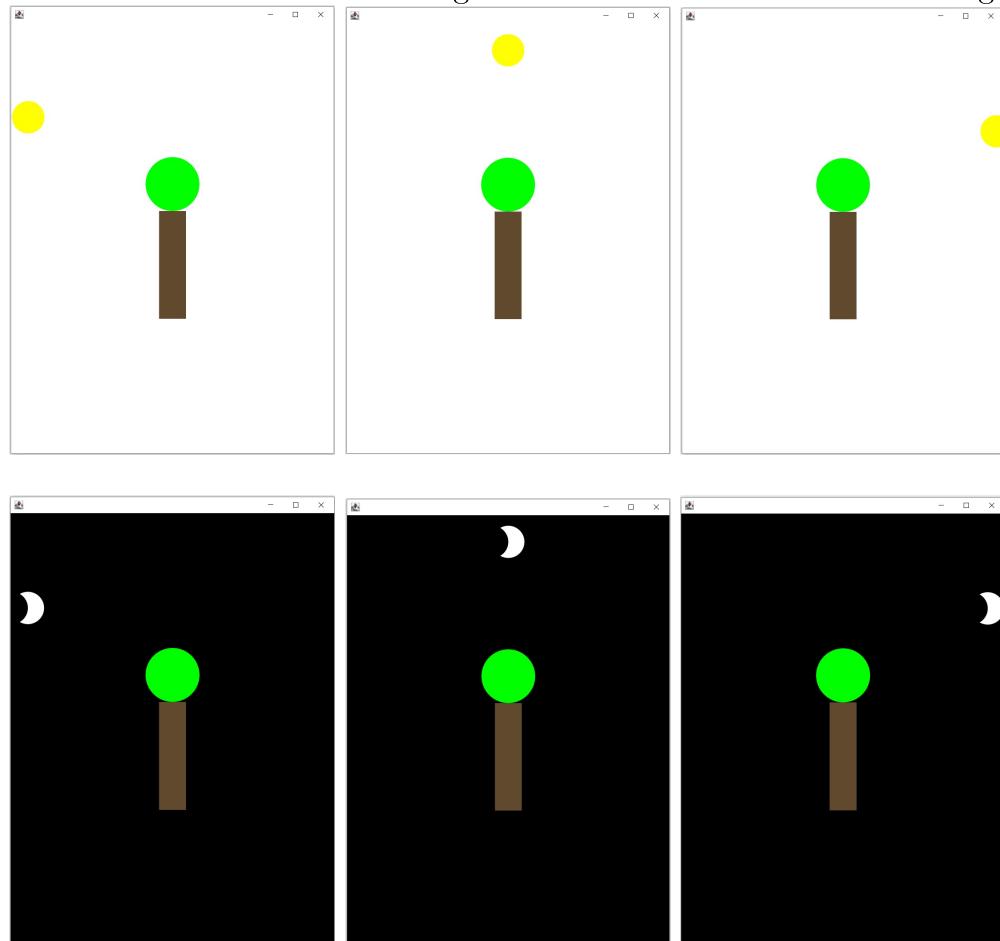
Passen Sie Ihre Animation dahingehend an, dass der Kreis an den Rändern der Leinwand abprallt. Die Breite bzw. Höhe der Leinwand können mit den Methoden `getLeinwandBreite()` bzw. `getLeinwandHoehe()` ausgewertet werden. Die Übergabeparameter für die Methode verschieben müssen in entsprechenden Variablen gespeichert werden. Überlegen Sie sich, wie sie beispielsweise bei einem Kreis mithilfe des Mittelpunkts und des Radius eine Kollision an den Rändern erkennen. Bei einer Kollision muss das Vorzeichen der entsprechenden Variable für die Verschiebung in x- oder y-Richtung, abhängig davon bei welchem Rand es zur Kollision kommt, umgedreht werden.

Achten Sie auch darauf, dass beim Verändern des Leinwandfensters während der Animation, der Kreis nicht verschwindet.

Erstellen Sie eine analoge Animation mit einem Rechteck.

**iii. Sehr anspruchsvoll:**

Passen Sie Ihre erste Zeichnung (Baum mit Sonne) so an, dass die Sonne aufgeht, sich im Bild auf einem Kreis bewegt und anschließend untergeht. Nach dem Sonnenuntergang soll es entsprechend Nacht werden und anstelle der Sonne soll sich ein sichelförmiger Mond im Bild auf einem Kreis bewegen.



## 5 Zugriffsmodifikatoren bzw. Sichtbarkeiten

### Aufgabe:

1. Erstellen Sie in einer Testklasse SichtbarkeitenTest ein Objekt kreis1 der Klasse Kreis und setzen Sie den Radius auf -1.

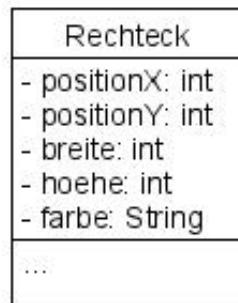
Lassen Sie das Objekt kreis1 mit der Methode zeichne eines Objekts der Klasse Leinwand zeichnen. Was fällt Ihnen auf?

Um obige Situationen zu vermeiden, müssen die meisten Attribute vor unkontrollierten Zugriff geschützt werden. Dies geschieht, indem Sie vor den Attributen das Schlüsselwort **private** schreiben:

```
public class Rechteck{
    private int positionX, positionY, breite, hoehe;
    private String farbe;
}
```

Dadurch ändert sich die so genannte **Sichtbarkeit** der Attribute. Auf **private**-Attribute kann nicht von anderen Klassen mit dem Punktoperator direkt zugegriffen werden bzw. sie sind für andere Klassen nicht sichtbar.

**Private-Sichtbarkeit** wird in UML mit einem – gekennzeichnet:



Wird in Java keine Sichtbarkeit explizit angegeben, so wird die so genannte Paketsichtbarkeit verwendet. In folgender Tabelle finden Sie eine Übersicht über die verschiedenen Sichtbarkeiten in Java.

Sichtbarkeit	Symbol (UML)	Sichtbar in eigener Klasse	Sichtbar für Klassen im gleichen Paket	Sichtbar für Klassen in anderem Paket
public	+	Ja	Ja	Ja
private	-	Ja	Nein	Nein
paketsichtbar	~	Ja	Ja	Nein

Die weitere Sichtbarkeit **protected** wird zu einem späteren Zeitpunkt behandelt.

Diese Sichtbarkeiten gelten auch für Methoden. Die bisherigen Methoden der Klasse Rechteck sollen alle anderen Klassen (unabhängig davon in welchem Paket sie liegen) nutzen können. So müssen die Methoden die Sichtbarkeit `public` besitzen:

Rechteck
<ul style="list-style-type: none"> <li>- <code>positionX: int</code></li> <li>- <code>positionY: int</code></li> <li>- <code>breite: int</code></li> <li>- <code>hoehe: int</code></li> <li>- <code>farbe: String</code></li> </ul>
<ul style="list-style-type: none"> <li>+ <code>Rechteck()</code></li> <li>+ <code>Rechteck(positionX:int, positionY:int, breite:int, hoehe:int, farbe:String)</code></li> <li>+ <code>verschieben(xRichtung:int, yRichtung:int): void</code></li> <li>+ <code>flaecheninhaltBerechnen(): double</code></li> <li>+ <code>hoeheBreiteVertauschen(): void</code></li> <li>+ <code>vergroessern(faktor:int): void</code></li> <li>+ <code>istQuadrat(): boolean</code></li> <li>+ <code>umfangBerechnen(): double</code></li> </ul>

Um nun auf `private`-Attribute immer noch zugreifen zu können, benötigt man so genannte **set-** bzw. **get-**Methoden in der Klasse Rechteck:

```
public void setPositionX(int positionX) {  
  
}  
  
public int getPositionX() {  
  
}
```

Mit diesen Methoden kann für jedes Attribut individuell der lesende bzw. schreibende Zugriff geregelt werden.

### Aufgaben:

- Ändern Sie die Sichtbarkeit der Attribute der Klassen `Rechteck` und `Kreis` auf `private`. In der Testklasse `SichtbarkeitenTest` sollte nun ein Compiler-Fehler auftreten. Erklären Sie kurz warum.

2. Legen Sie für jedes Attribut in der Klasse Rechteck und Kreis eine set- bzw. get-Methode an. Da dies bei vielen Attributen sehr viel (stupide) Arbeit bedeuten würde, kann Eclipse diese Methode automatisch generieren:  
Menüreiter Source → „Generate Getters and Setters“  
**Hinweis:** Sollten Sie die Methoden händisch anlegen, achten Sie darauf, dass die get- bzw. set- Methoden wie nach obigen Muster benannt werden: set/get[Attributname mit groß geschriebenem Anfangsbuchstaben]
  3. Passen Sie nun Ihre Klassen Rechteck und Kreis so an, dass keine negativen Werte mehr für hoehe, breite und radius möglich sind. Sollte man versuchen, diesen Attributen einen negativen Wert zuzuweisen, so soll eine entsprechende Ausgabe auf der Konsole erscheinen.  
(Hinweis: i f)