PGdP

Besprechung Quiz

Parameter

Objektorientierung

Listen

P-Aufgaben

Folien: go.tum.de/904005

Tobias Schamel

Besprechung Quiz

Ausführliche Erklärung auf den Folien der letzten Woche im Kapitel "Datentypen".

Parameter

Call-by-Value: Methodenaufruf übergibt eine Kopie des Wertes. Bei Referenzwerten wird eine Kopie der Referenz übergeben.

```
int[] array = {1};  int zahl = 2;
public static void changeWert(int[] a, int b) {
   a[0] = 0;
   b = 0;
}
changeWert(array, zahl);
write("Array: " + array[0] + ", Zahl: " + zahl;
```

Übergeben wird der Zeiger auf das Array

Übergeben wird der Wert des Integers.

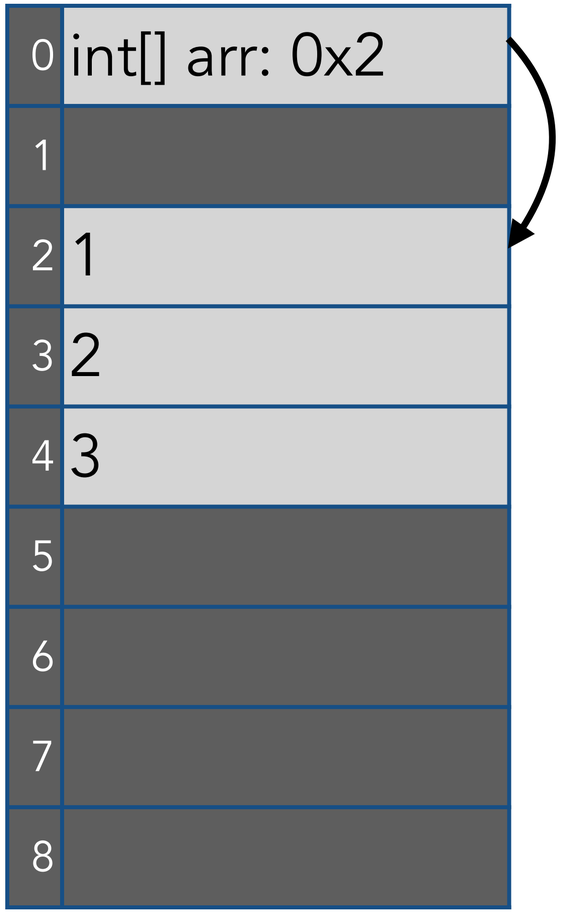
Parameter

Recap: Array/Referenzwerte

```
1   int[] arr;
2
3   arr = new int[3];
4
5   for(int i = 0; i < 3; i++)
6     arr[i] = i+1;
7   write(arr[1] + arr[2]);
8   int[] a2 = arr;
9   a2[1] = 7;
10  write(arr[1] + arr[2]);
```

```
> 5
```

Speicher



Objektorientierung

Objektorientierung, Bibliothek:

In der Bibliotheksverwaltung werden alle Bücher mit ihrem Autor und ihrer ISBN, sowie ihrer Seitenzahl gespeichert. ~~Zusätzlich wird für jedes ausgeliehene Buch ein Kunde mit Namen und Geburtsjahr gespeichert. Für jedes Buch ist zusätzlich eine Bewertung zwischen 1 und 5 hinterlegt.~~
Für jeden Autor sind alle Bücher, sein Name und sein Geburtsjahr gespeichert, die er geschrieben hat.

Objektorientierung

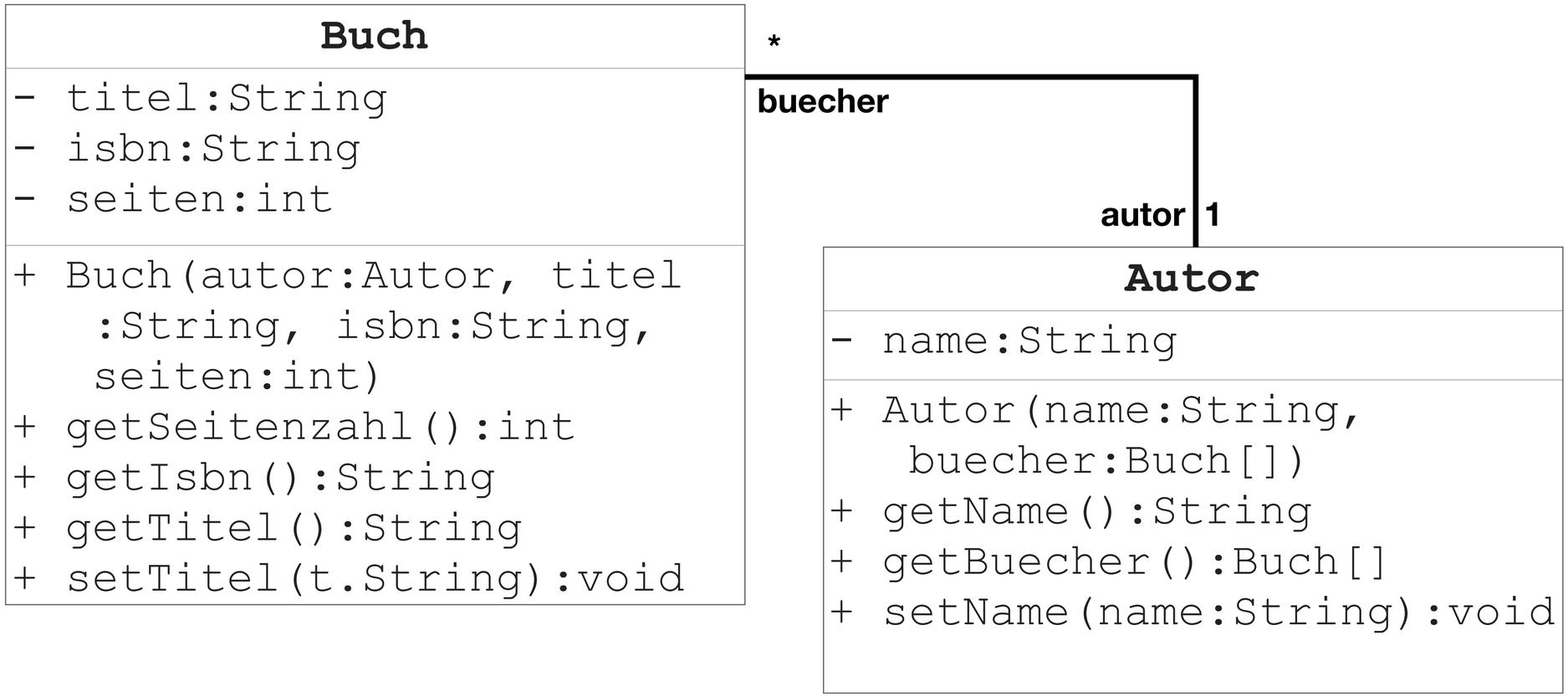
Objektorientierung, Bibliothek:

In der Bibliotheksverwaltung werden alle Bücher mit ihrem Autor und ihrer ISBN, sowie ihrer Seitenzahl gespeichert. Für jeden Autor sind Name und alle Bücher, die er geschrieben hat gespeichert.

Wir versuchen die reale Welt aus Objekten & Gegenständen mit Code nachzubauen.

Objektorientierung

Objektorientierung, Bibliothek:



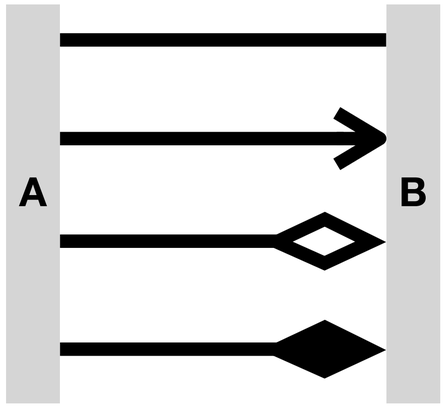
Objektorientierung

Exkurs UML

Klassenname
Attribute
Konstruktor (opt.)
Methoden

Modifier	Klasse	Unterklasse	Package	Welt
+public	✓	✓	✓	✓
-private	✓	✗	✗	✗
#protec.	✓	✓	✓	✗

Methoden mit :Rückgabetyp, Attribute mit :Typ



Bidirektionale Assoziation: A kennt B & B kennt A

Unidirektionale Assoziation: A kennt B

Aggregation: A ist Teil von B, *B kennt A*

Komposition: wie Aggregation, A kann nicht ohne B existieren

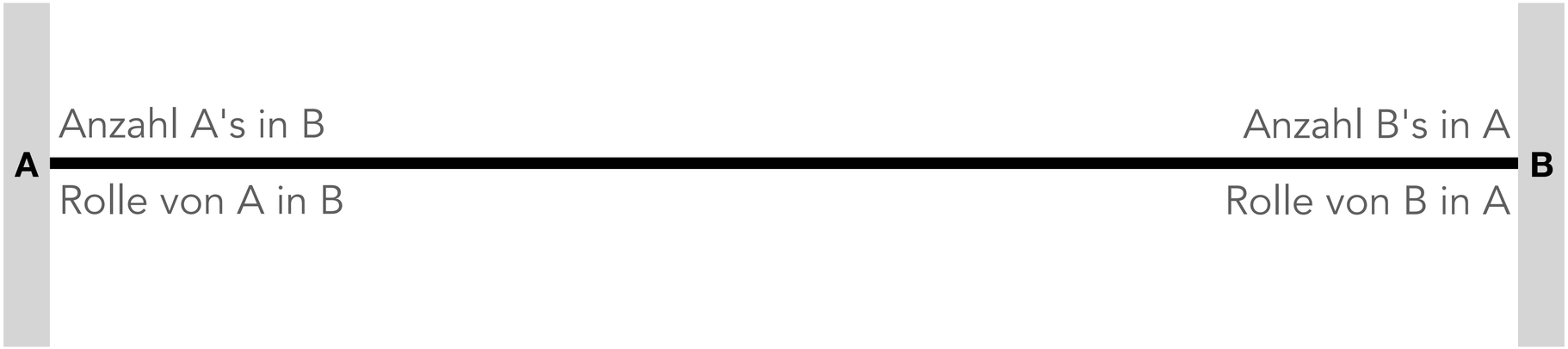
Objektorientierung

Exkurs UML

Klassenname
Attribute
Konstruktor (opt.)
Methoden

Multiplizitäten

1	exakt 1
*****	mehrere, inkl. 0



Objektorientierung

Klassen, Attribute

```
public class Buch {
   //Attribute
   private int seitenzahl;
   private String titel;

   //Konstruktor
   //Methoden
}
```

Attribute sollten private sein: Datenkapselung, Plausibilität sichern

Objektorientierung

Klassen, Konstruktor

```
public class Buch {
   //Attribute
   //Konstruktor
   public Buch(int seiten, String titel) {
      seitenzahl = seiten;
      this.titel = titel;
   }

   //Methoden
}
```

Standardkonstruktor ohne Attribute automatisch vorhanden; wird überschrieben.

Objektorientierung

Klassen, Konstruktor

```
public class Buch {
   //Attribute
   //Konstruktor
   //Methoden
   public static void main(String[] args) {
      Buch b1 = new Buch(944, "Wohlstand der
        Nationen")
   }
}
```

Dient dem Erstellen von Objekten.

Objektorientierung

Klassen, Methoden

```
public class Buch {
   //Attribute
   //Konstruktor
   //Methoden
   public void setSeitenzahl(int seitenzahl) {
      //prüfen, ob Seitenzahl Sinn ergibt
      if(seitenzahl > 0)
         this.seitenzahl = seitenzahl;
   }
}
```

Objektorientierung

Klassen, Methoden

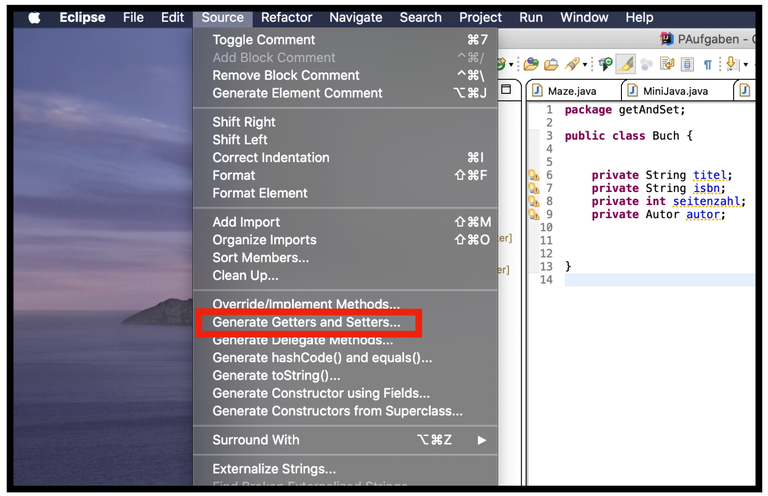
```
public class Buch {
   //Attribute
   //Konstruktor
   //Methoden
   public void setSeitenzahl(int seitenzahl) {
      //prüfen, ob Seitenzahl Sinn ergibt
      if(seitenzahl > 0)
         this.seitenzahl = seitenzahl;
   }
}
```

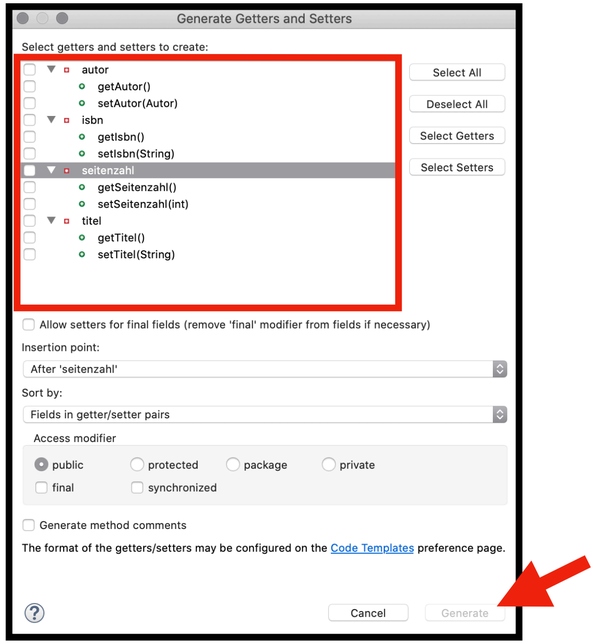
Mit dem Wort **this** referenziert man auf das Objekt selbst.

Objektorientierung

Getter und Setter

- Attribute sind üblicherweise private! (z.B. damit man beim 'setten' Plausibilitätsprüfungen durchführen kann.)





Objektorientierung

toString

- Jede Klasse sollte über eine toString() Methoden verfügen
- Ermöglicht Ausgabe von Infos auf Konsole

```
public String toString() {
   return "Buch: " + this.titel + " mit "
    + this.seitenzahl + " Seiten";
}
```

Objektorientierung

== für primitive Datentypen, equals() Methode für komplexe Datentypen.

Gleichheit und Identität

- Der Vergleichsoperator == prüft auf Identität
  > das selbe
- Die Methode equals(Object obj):boolean prüft auf Gleichheit
  > das gleiche

```
public boolean equals(Buch b) {
  return (b.titel == this.titel &&
    b.autor.equals(this.autor));
}
```

Objektorientierung

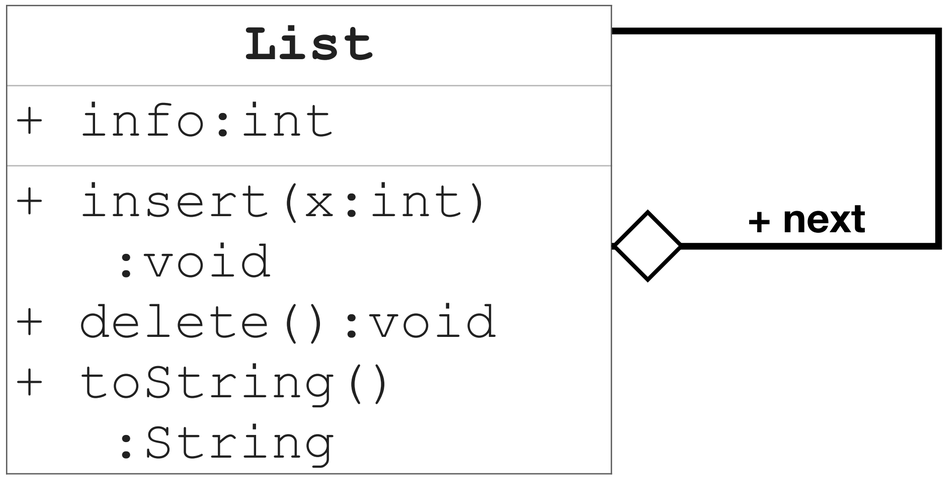
Objekte und Klassen, Unterschiede

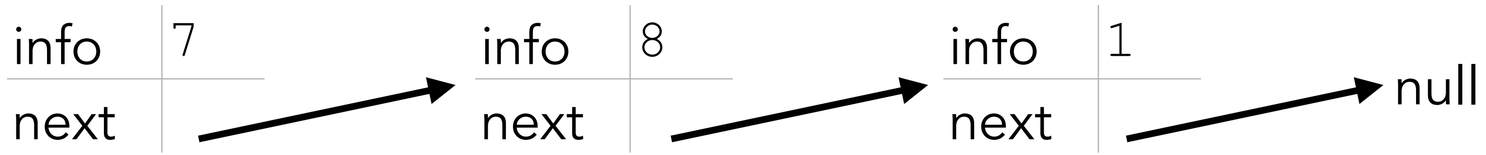
- Klassen sind Abstraktionen, Objekte sind konkrete Ausprägungen der Abstraktionen.

Klassen (static)	**Objekt**
Methoden:	Methoden:
- können ohne Instanz (Objekt) aufgerufen werden	- benötigen Instanz und können auf Objektattribute zugreifen
Attribute:	Attribute:
- sind für alle Objekte gleich	- sind eindeutig für jedes Objekt
`classname.method();`	`objname.method();`
`classname.varname;`	`objname.attrname;`

Listen

Idee, Selbstreferenz



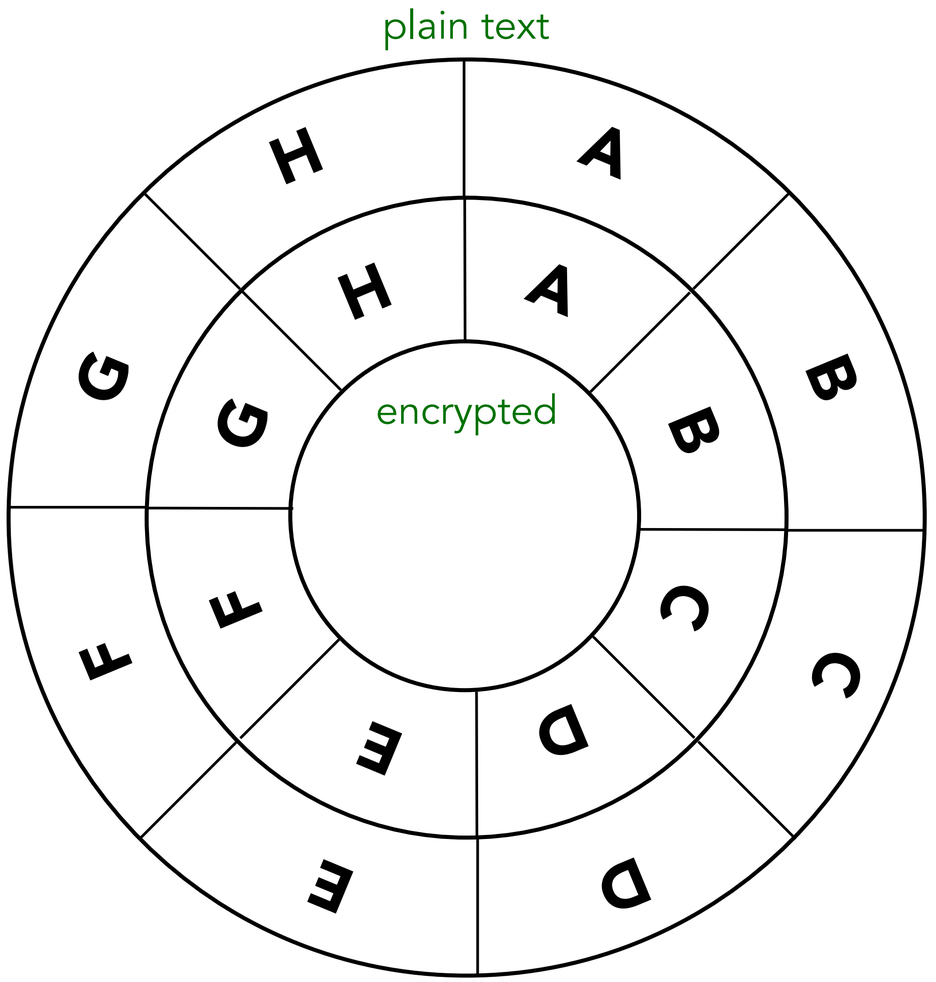


P06.01

Cäsar Chiffre

BACHE

DACH

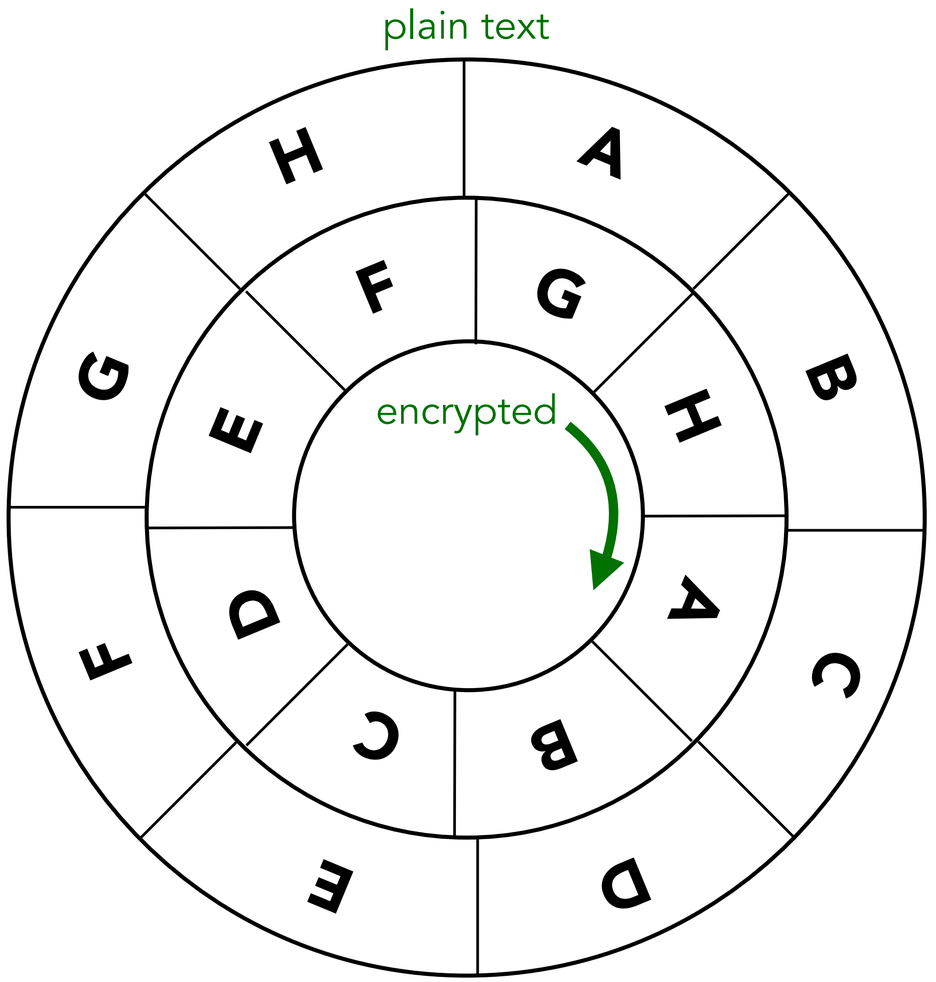


P06.01

Cäsar Chiffre

`HGAFC`

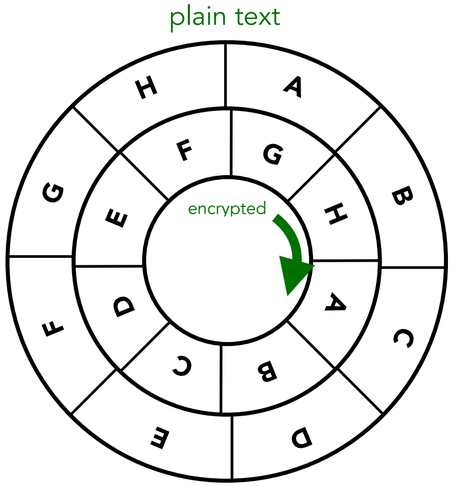
`BGAF`



P06.01

Cäsar Chiffre

- Alle Zeichen werden um k im Alphabet verschoben




```
public static char[] toArray(String input)
  //String in Character-Array umwandeln
public static String encrypt(char[] input,
  int key)  //'encrypten' > verschieben um k
```

P06.01

Cäsar Chiffre: ASCII Tabelle

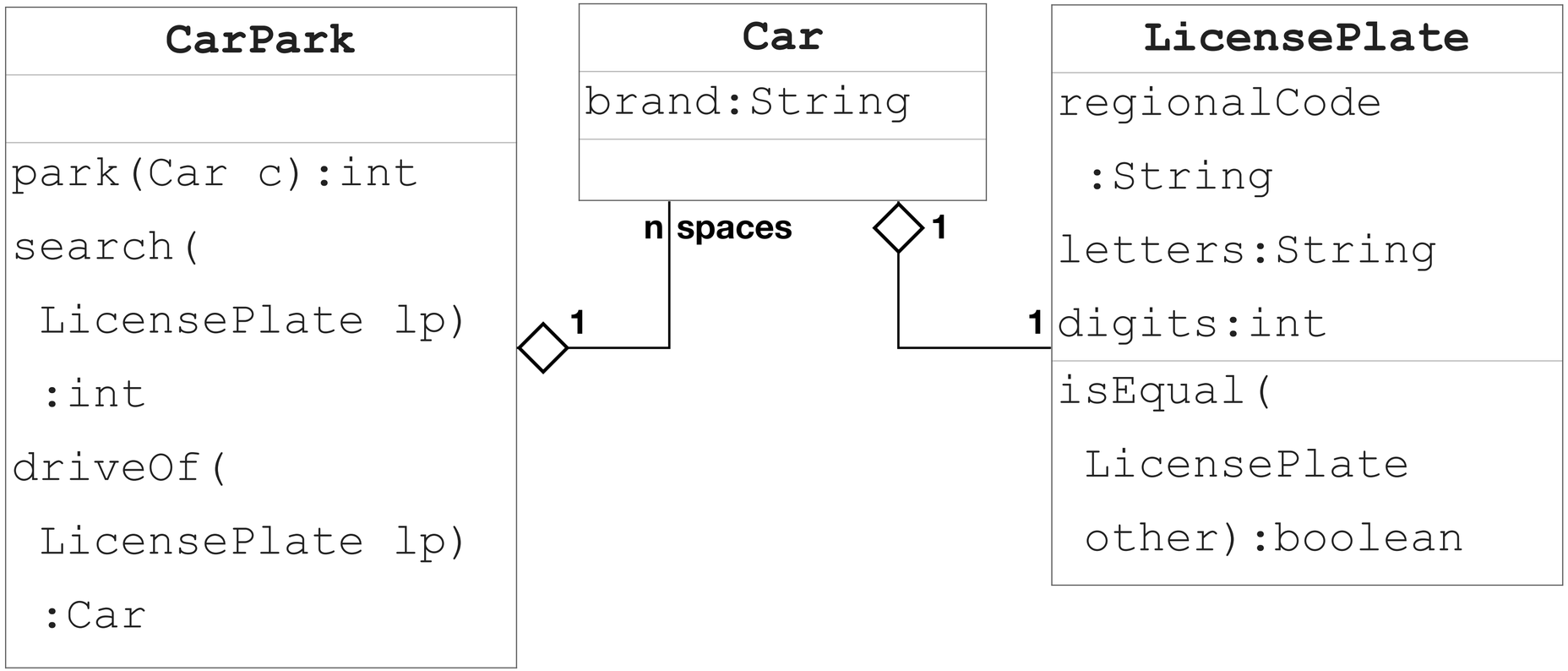
- **`char` ≡ `short`**

Dezimal	Zeichen	Binär	Hexadezimal	Dezimal	Zeichen	Binär	Hexadezimal	Dezimal	Zeichen	Binär	Hexadezimal	Dezimal	Zeichen	Binär	Hexadezimal
64	@	1000000	40	80	P	1010000	50	96	`	1100000	60	112	p	1110000	70
65	A	1000001	41	81	Q	1010001	51	97	a	1100001	61	113	q	1110001	71
66	B	1000010	42	82	R	1010010	52	98	b	1100010	62	114	r	1110010	72
67	C	1000011	43	83	S	1010011	53	99	c	1100011	63	115	s	1110011	73
68	D	1000100	44	84	T	1010100	54	100	d	1100100	64	116	t	1110100	74
69	E	1000101	45	85	U	1010101	55	101	e	1100101	65	117	u	1110101	75
70	F	1000110	46	86	V	1010110	56	102	f	1100110	66	118	v	1110110	76
71	G	1000111	47	87	W	1010111	57	103	g	1100111	67	119	w	1110111	77
72	H	1001000	48	88	X	1011000	58	104	h	1101000	68	120	x	1111000	78
73	I	1001001	49	89	Y	1011001	59	105	i	1101001	69	121	y	1111001	79
74	J	1001010	4A	90	Z	1011010	5A	106	j	1101010	6A	122	z	1111010	7A
75	K	1001011	4B	91	[	1011011	5B	107	k	1101011	6B	123	{	1111011	7B
76	L	1001100	4C	92	\	1011100	5C	108	l	1101100	6C	124	|	1111100	7C
77	M	1001101	4D	93	]	1011101	5D	109	m	1101101	6D	125	}	1111101	7D
78	N	1001110	4E	94	^	1011110	5E	110	n	1101110	6E	126	~	1111110	7E
79	O	1001111	4F	95	_	1011111	5F	111	o	1101111	6F	127		1111111	7F

Quelle: virtualuniversity.ch

P06.02

Parkhaus



P06.03

Datenkapselung

```
public class
 Rabbit {
  public int age;

  public static void main(String[] args) {
    Rabbit rabbit = new Rabbit();
    rabbit.age = -5;
  }
}
```

Modifier	Klasse	Unterklasse	Package	Welt
+public	✓	✓	✓	✓
-private	✓	✗	✗	✗
#protec.	✓	✓	✓	✗

P06.03

Datenkapselung

```
public class Rabbit {
    private int age;
    public void setAge(int age) {
        if (age >= 0)
            this.age = age;
    }
    public static void main(String[] args) {
        Rabbit rabbit = new Rabbit();
        rabbit.setAge(-5);
    }
}
```

P06.04

Doppel Verkettete Listen