

Zeichen und Zeichenketten

- Datentyp **char**
 - Primitiver Datentyp (wie **int**, **double**, ...), also keine Klasse
 - Speichert ein einzelnes Zeichen (16 Bit, basierend auf Unicode-Standard)
 - Verwendung: **char** **c** = **'A'**;
 - Wichtig: Verwendung einfacher Anführungszeichen (Hochkomma (')), keine Akzente (' , ´))

Operationen auf Zeichen

- Da **char** ein primitiver Datentyp ist, kann er keine eigenen Operationen bereitstellen
- Zur Klassenbibliothek von Java gehört die Klasse **Character**, die nützliche Methoden zur Arbeit mit einzelnen Zeichen bereitstellt, z.B.:

static boolean isDefined(char c) Prüft, ob das in **c** gespeicherte Zeichen in Unicode definiert ist

static boolean isLetter(char c) Prüft, ob das in **c** gespeicherte Zeichen ein Buchstabe ist

static boolean isDigit(char c) Prüft, ob das in **c** gespeicherte Zeichen eine Ziffer ist

static char toLowerCase(char c) Konvertiert das in **c** gespeicherte Zeichen in einen Kleinbuchstaben

static char toUpperCase(char c) Konvertiert das in **c** gespeicherte Zeichen in einen Großbuchstaben

- Schreibe (auf Papier oder in einem Texteditor) Code, der eine Variable vom Typ **char** prüft, ob sie ein Buchstabe ist. Falls ja, soll der Text "Buchstabe" ausgegeben werden, ansonsten der Text "Kein Buchstabe".

- Datentyp **String**
 - Klasse aus der Klassenbibliothek von Java
 - Speichert eine Folge von einzelnen Zeichen als Objekt
 - Verwendung: **String** s = "Hallo Welt!"
 - Es gibt noch viele weitere Konstruktoren für Objekte vom Typ **String**
 - Wichtig: Verwendung doppelter Anführungszeichen (")
 - Strings ins Java sind **unveränderbar** (immutable), d. h. sie können nach der Erzeugung nicht mehr geändert werden
 - Konsequenz: Will man an einem String etwas ändern, so muss man dafür einen neuen erzeugen

+-Operator Verknüpft zwei String zu einem neuen, z. B.

```
String text = "Hallo " + "Welt!";
```

char charAt(int index) Gibt das Zeichen (als Wert vom Typ **char**) an der Position **index** zurück, z. B.

```
char c = "Hallo".charAt(1);
```

Vorsicht: Wie bei Arrays beginnt die Zählung bei 0

int length() Gibt die Länge eines Strings zurück, z. B.

```
int l = text.length();
```

- Schreibe eine Methode **verknüpfe**, die zwei Strings als Parameter erwartet
 - Die Methode soll beide Strings zu einem neuen zusammenfügen
 - Dieser neue String soll ausgegeben werden
- Verändere die Methode so, dass der neu erzeugte String ein Leerzeichen zwischen den beiden eingegebenen Strings beinhaltet

- Schreibe eine Methode **buchstabiere**, die einen String als Parameter erwartet
 - Die Methode soll die Zeichen des Strings nacheinander einzeln ausgeben, jeweils gefolgt von einem “-”
- Zusatzaufgaben:
 - Verändere die Methode so, dass nach dem letzten Zeichen kein “-” mehr ausgegeben wird
 - Modifiziere die Methode so, dass sie stattdessen rückwärts buchstabiert

String toUpperCase() Gibt einen neuen String zurück, bei dem alle Zeichen in Großbuchstaben umgewandelt wurden, z. B.

```
String g = text.toUpperCase();
```

String toLowerCase() Gibt einen neuen String zurück, bei dem alle Zeichen in Kleinbuchstaben umgewandelt wurden, z. B.

```
String k = g.toLowerCase();
```

Vorsicht: Beide Methoden geben einen neuen String zurück, der ursprüngliche bleibt unverändert!

Ein Palindrom ist ein Wort, das vorwärts und rückwärts gelesen identisch ist, also z. B. RENTNER oder LAGERREGAL.

- Schreibe eine Methode **istPalindrom**, die einen String als Parameter erwartet und prüft, ob dieser String ein Palindrom ist
 - Die Methode soll den String zunächst in Großbuchstaben umwandeln und dann den umgewandelten String prüfen
 - Die Methode soll ein Ergebnis vom Typ **boolean** zurückliefern

Was zeichnet ein gutes Passwort aus?

- Vernünftige Länge (10 Zeichen oder mehr)
- Mischung aus Buchstaben, Zahlen, Sonderzeichen
- Kein erkennbarer Bezug zur Person (Geburtsdatum, Name der Katze etc.)
- Kein einzelnes Wort (Wörterbuchattacken)

Aufgabe 5

- Schreibe eine Methode `boolean istPasswortGut(String pw)`, die das im Parameter `pw` übergebene Passwort auf folgende Kriterien prüft:
 - Mindestlänge von 10 Zeichen
 - Mindestens ein Groß- und ein Kleinbuchstabe vorhanden
 - Mindestens eine Ziffer vorhanden
 - Mindestens ein Sonderzeichen vorhanden
- Wenn **alle** Kriterien erfüllt sind, soll die Methode **true** zurückgeben, ansonsten **false**
- Zusatzaufgabe:
 - Erweitere die Methode so, dass sie ausgibt, welche Kriterien nicht erfüllt wurden

- Statt eines komplizierten Passworts mit Ziffern, Sonderzeichen etc. erfüllt auch ein langes Passwort aus zufällig gewählten Wörtern in der Regel gute Zwecke, siehe hier ;-)
- Das Erzwingen von bestimmten Zeichentypen hat auch Nachteile, denn ein Angreifer weiß dann, dass alle Passwörter, die die entsprechenden Zeichentypen nicht enthalten, gar nicht ausprobiert werden müssen
- Die häufigsten Passwörter weltweit 2024¹:
 1. 123456
 2. 123456789
 3. 12345678

¹<https://nordpass.com/de/most-common-passwords-list/>

Vergleichen von Strings (1)

- Welche Ausgabe liefert der folgende Code?

```
String s1 = new String("Hallo Welt");  
String s2 = new String("Hallo Welt");  
  
if ( s1 == s2 ) {  
    System.out.println("Gleich");  
}  
else {  
    System.out.println("Ungleich");  
}
```

- Der Operator `==` testet nur die Referenzen `s1` und `s2` auf Gleichheit, nicht die Inhalte der Zeichenketten!
 - Ein Vergleich zwischen identischen Zeichenketten, die an verschiedenen Stellen im Speicher stehen, liefert das Ergebnis **false**

Vergleichen von Strings (2)

- Um Strings inhaltlich zu vergleichen, können die Methode `equals` und `equalsIgnoreCase` der Klasse `String` benutzt werden, z.B.:

```
if ( s1.equals(s2) ) { ... }
```

 - Die Methoden geben bei inhaltsgleichen Strings `true` zurück, ansonsten `false` (wobei `equalsIgnoreCase` Groß- und Kleinschreibung ignoriert)
- Alternativ können auch die Methoden `compareTo` und `compareToIgnoreCase` der Klasse `String` verwendet werden, z.B.:

```
if ( s1.compareTo(s2) == 0 ) { ... }
```

 - Der Vergleich erfolgt lexikographisch, d. h. die Methode gibt zurück:
 - `0` falls beide Strings übereinstimmen
 - einen Wert `< 0` falls `s1` bei alphabetischer Sortierung **vor** `s2` kommt
 - einen Wert `> 0` falls `s1` bei alphabetischer Sortierung **nach** `s2` kommt