Technische Hochschule **Rosenheim**
Technical University of Applied Sciences

Programming Basics – WiSe21/22

Characters and Strings

Prof. Dr Silke Lechner-Greite

Table of contents – planned topics

Chapter 9: Characters and Strings

Character encoding

Aim: enabling the transfer of data and exchange of documents within different countries

ASCII

- 7-bit encoding
- Contains 128 characters
- Code point: each character has a unique position
- Basis for subsequent standards
- 1963

ISO/IEC 8859-1

- 8-bit encoding
- Space for 256 characters
- Contains 197 characters → empty slots
- Latin-1: extended by the Latin script
- Extended ASCII, ASCII code points are retained
- Without control characters
- 1985

ISO 8859-1

- Extension of Latin-1 with control characters at code points 0-31 and 127
- Contains complete ASCII
- Fills additional empty spaces with additional special characters.

Unicode

- Aim: encoding every character in the world
- Unicode includes ISO 8859-1 → easier file conversion
- Describes over 100,000 characters
- Representation: Hexadecimal with the prefix U+ followed by hexadecimal numbers
- A : U+0041
- Java 8 supports Unicode 6.2

Character sets

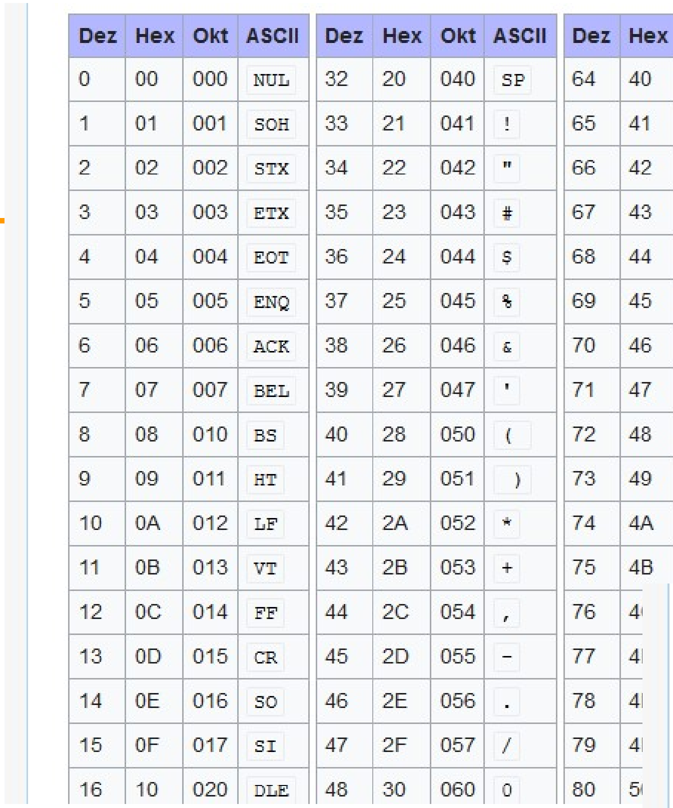
- Character set = table of characters <-> code point
- Different character sets

Character Set	Range	Description
ACSII	0-127	Aimed at US applications; no German umlauts and other special characters from European languages.
Latin-1 ISO-8859-1	0-255	0-127 = ASCII 128-255: predominantly characters from west European languages
Latin-15	0-255	Like Latin-1, except code 164 = currency symbol €
Unicode	0-100000+	0-127 = ASCII 0-255 = Latin-1 256+:Zeichen von Weltsprachen

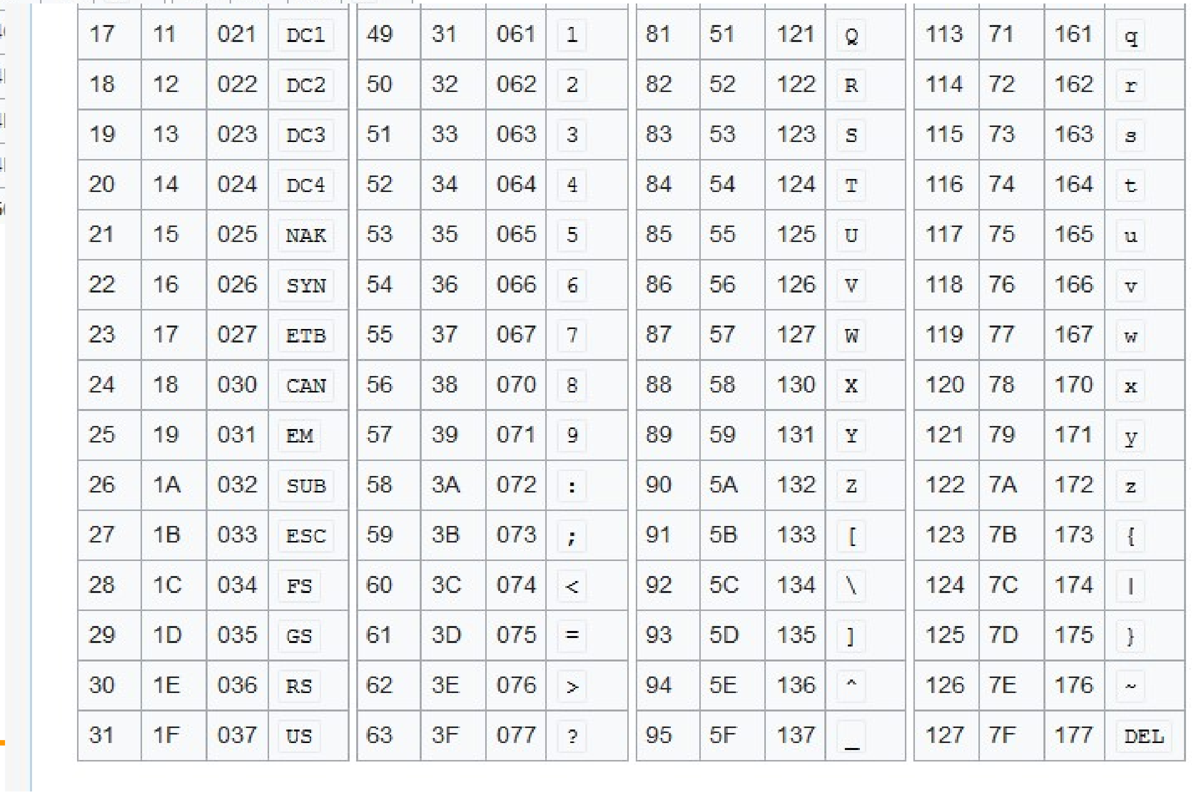
- Java uses Unicode: https://home.unicode.org/

Translated from source: Schiederrmeier

Character code = unique `int` value for each character



Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII
0	00	000	NUL	32	20	040	SP	64	40	100	@	96	60	140	`
1	01	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	02	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	03	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	04	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	05	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	06	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	07	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	08	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	09	011	HT	41	29	051	)	73	49	111	I	105	69	151	i
10	0A	012	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	VT	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	FF	44	2C	054	,	76	4						
13	0D	015	CR	45	2D	055	-	77	4						
14	0E	016	SO	46	2E	056	.	78	4						
15	0F	017	SI	47	2F	057	/	79	4						
16	10	020	DLE	48	30	060	0	80	5						



17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

More details:
https://en.wikipedia.org/wiki/ASCII

Graphs taken from:
https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

Type `char`

- Individual text characters are represented by the primitive type `char` (*character*)
- Representation of characters using 16 bits
- `char` literals: Characters in inverted commas *(quotation marks)*

  Examples:

Literal	Meaning
'a'	lower case letter “a”
'8'	number “8”, not `int` value 8
'%'	percent sign
' '	space

- Escape sequences: saves typing complicated Unicodes
  - Newline: '\n' Single quote: '\' '
  - Tabulator: '\t' Double quote: '\" '

Operations with `char`

- Declaration and assignment of a variable:

```
char letter;
letter = 'a';
```

- Comparison of characters for equality and inequality:

```
char seven = '7';
if (seven == 'a')        //false
     ...
```

- Size comparison for lower case or upper case letters according to the alphabet:

```
char capitalA = 'A';
if (capitalA < 'B')          //true
     ...
else if (capitalA > 'B')   //false
          …
```

Arithmetic with `char` values

- Increment/decrement of `char` variables leads to the "next" character

- Example:

```
for(char letter = 'Z'; letter >= 'A'; letter--)
  System.out.print(letter);
```

Library methods for characters

- Static methods of the `Character` class
- Test methods – shown with prefix "is"
- Selection:

Let‘s look at the API about Character class and the definition of whitespaces

`boolean isLetter(char ch)`	is ch a letter (upper or lower case?)
`boolean isDigit(char ch)`	is ch a digit
`boolean isWhitespace(char ch)`	is ch a space character (whitespace)
`boolean isLowerCase(char ch)`	is ch a lower-case letter
`boolean isUpperCase(char ch)`	is ch an upper-case letter
`char toLowerCase(char ch)`	small letter to ch if it exists; ch otherwise
`char toUpperCase(char ch)`	capital letter to ch if it exists; ch otherwise

- More:
  - https://docs.oracle.com/javase/8/docs/api/java/lang/Character.html
  - https://docs.oracle.com/javase/7/docs/api/java/lang/Character.html#isWhitespace(int)

Exercise – Library methods for characters

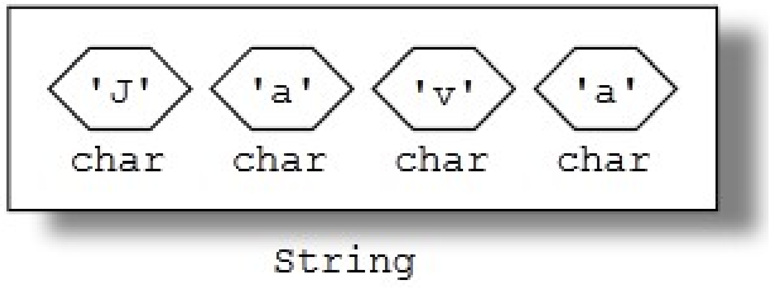
- Live exercise
  - Complete Task 1 on the live exercises sheet “Characters and Strings”
  - You have 15 minutes.

Chapter 9: Characters and Strings

Type `String`

- Text /strings are represented by the type `String`
- Container type: stores items of other types
  - Individual elements of strings: `char`
  - Any number of `char` elements

- Structure of a string:



String literals

- In order to work with strings, a string object must exist.
- String literals: text between double quotation marks → is automatically a string object
- Therefore, a method call can be executed with . after a string literal.
- Examples:

```
"Java"
"I am a string!"
" "
""
```

All character representations are allowed

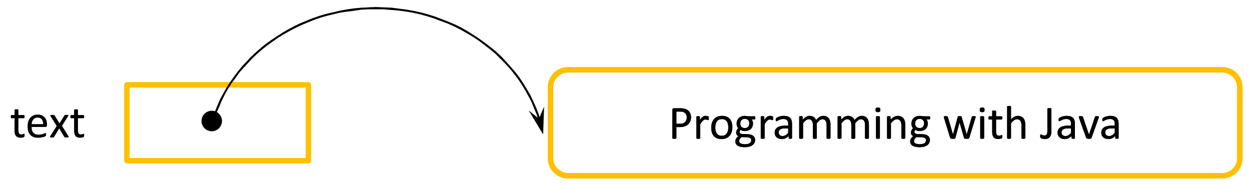
```
"'a'"
"two\n\lines"
"M\u00FCnchen"
```

- Length of a string = number of characters

Creating a string object

- Create and reference to the string object:

```
String text;
text = new String("Programming with Java");
```



- Short form (and preferred version):

```
String text = "Programming with Java";
```

Special feature of `strings`

- Strings are unchangeable (immutable)
  - Insertion, replacement or removal of individual characters is not possible
- Note: `String` variables are changeable, except they are defined as `final`

```
String text;
text = "Java";                      // ok
text = "Compiler";                  // ok

final String fixed;
fixed = "I never change";    // ok
fixed = "error";             // error because of final
```

Concatenation

- Concatenation: + with strings
  - Chaining (concatenation) with the + operator
  - Produces a new, third string from the operands
- Example:

```
"Java" + "compiler" → "Javacompiler"
```

- Different data types and one of them is a string: the other data types are implicitly type casted to a String followed by a concatenation
- Polymorphism:

```
3 + 5 → 8               int
3.0 + 5.0 → 8.0         double
"3" + "5" → "35"        string
```

`String` class

- Documentation of the class: http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

- Selection:

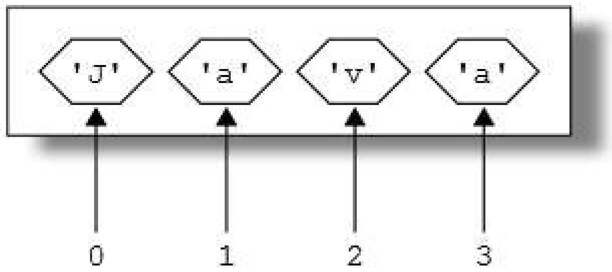
```
// constructors
public String();
public String(String value);

// methods
public char charAt(int  index);
public int length();
public boolean equals(Object  anObject);
public String toLowerCase();
public String toUpperCase();
public String substring(int  beginIndex);
public String substring(int  beginIndex, int endIndex);
public int compareTo(String anotherString)

...
```

(Object) methods of the `String` class (1)

- `charAt` copies individual characters from the string
  - Argument: index of the desired character
    - Acceptable values: 0, 1, ..., string length -1



    - Index <0 or index >= string length: index error, programme termination

- Examples:

```
String s = "Java";
char c;
c = s.charAt(1);                // c = 'a'
c = s.charAt(-1);               // error
c = s.charAt(4);                // error
```

(Object) methods of the `String` class (2)

- `length()` returns the number of characters
  - Result never negative, 0 for empty string
  - Length conceptually unlimited

- Examples:
  - Determining the length of a string

```
String s = "Java";
int len = s.length();                    // len = 4
```

  - Outputting all characters of a string

```
for(int i = 0;  i < s.length();  i++)
    System.out.println(s.charAt(i));
```

(Object) methods of the `String` class (3)

- Comparison of strings (test for equal contents)
  - `boolean equals(Object anObject)`
  - Compares character-by-character and returns `true` if equal

```
String      s1 = "Hello";
String      s3 = new String("Hello");
boolean b3 = s3.equals(s1); // b3 is true
```

⚠ Operator `==` checks the identity of `String` objects, not the contents!

(Object) methods of the `String` class (4)

- Converting strings: changing upper case and lower case

  - `toLowerCase(): String`
    converts all letters of the string to lower case

  - `toUpperCase(): String`
    converts all letters of the string to upper case

- Examples:
  - `"Java compiler".toLowerCase() → "java compiler"`
  - `"Java compiler".toUpperCase() → "JAVA COMPILER"`

(Object) methods of the `String` class (5)

- Splitting strings
  - `String substring(int from)`
    Partial string (i.e. substring) from index `from` to the end of the string
  - `String substring(int from, int to)`
    Partial string (i.e. substring) from index `from` to (exclusively) index `to`
- Examples:
  - `"Java compiler".substring(2)` → `"va compiler"`
  - `"Java compiler".substring(2, 7)` → `"va co"`

(Object) methods of the `String` class (6)

- Lexicographical order
  - `int compareTo(String anotherString)`
  - Two strings are given: `s1 , s2`
  - Comparison using `s1.compareTo(s2)`
  - Result

< 0	if `s1` is alphabetically before `s2`
= 0	if `s1` equals `s2`
> 0	if `s1` is alphabetically after `s2`

- Examples:

```
"hello".compareTo("java")     → -2
"hello".compareTo("hello")    →  0
"hello".compareTo("compiler") →  5
```

Static methods of the `String` class

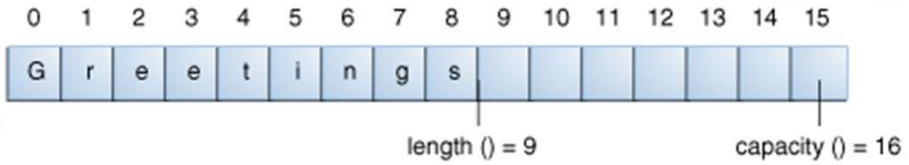
- Convert numbers and strings into one another
  - `static String valueOf(int i)`
  - `static String valueOf(long l)`
  - `static String valueOf(float f)`
  - `static String valueOf(double d)`
  - `. . .`

- Examples:
  - `String s1 = String.valueOf(99.9);`

`StringBuilder` class

- If the programme works extensively with strings, new string objects are created continuously as intermediate results
  => possible performance loss
- Switch to `StringBuilder` class
  - `StringBuilder` objects are like `String` objects, but they can be changed -> changeable strings
  - Within the `StringBuilder` class, strings are treated as arrays of `chars` that can change in length.
  - `StringBuilder` makes it possible to modify a string at any time in terms of length and content -> provides dedicated methods to achieve this
- Note: it is preferable to use strings, unless there is an advantage to using the `StringBuilder` , e.g. performance. Example: concatenation of long strings -> more efficient with `StringBuilder`

`StringBuilder` class



https://docs.oracle.com/javase/tutorial/java/data/buffers.html

- Constructors:
  - `StringBuilder():` empty StringBuilder object (capacity = 16)
  - `StringBuilder(CharSequence cs): StringBuilder` with cs as contents in the StringBuilder object + 16 empty elements
  - `StringBuilder(int initCapacity):` empty StringBuilder object with initCapacity elements
  - `StringBuilder(String s): StringBuilder` with cs as contents in the StringBuilder object + 16 empty elements
- Simple conversion:
  - `String -> StringBuilder`

```
String s = "Java";
StringBuilder b = new StringBuilder(s);
```

  - `StringBuilder -> String`

```
StringBuilder b = new StringBuilder("Java");
String s = b.toString();
```

Methods of `StringBuilder` (1)

- Initial situation:

```
StringBuilder b = new StringBuilder("Java compiler");
```

- `StringBuilder append(char c)`
  adds the character `c` at the end

```
b.append('#') → "Java compiler#"
```

- `StringBuilder insert(int at, char c)`
  inserts the character c at index at; the rest moves backwards

```
b.insert(4, '#') → "Java# compiler"
```

Methods of `StringBuilder` (2)

- `StringBuilder deleteCharAt(int at)`
  deletes the character at index `at`; the rest moves forwards

```
b.deleteCharAt(4) → "Javacompiler"
```

- `StringBuilder delete(int from, int to)`
  deletes the substring from index `from` to (exclusively) index `to;`
  the rest moves forwards

```
b.delete(4, 8) → "Javapiler"
```

⚠ Methods do not create a new `StringBuilder` object,
but modify **`this`**

https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html

Exercise – Applying methods

- Live exercise
  - Complete Task 2 on the
    live exercises sheet “Characters and strings”
  - You have 15 minutes.

`StringBuffer` class

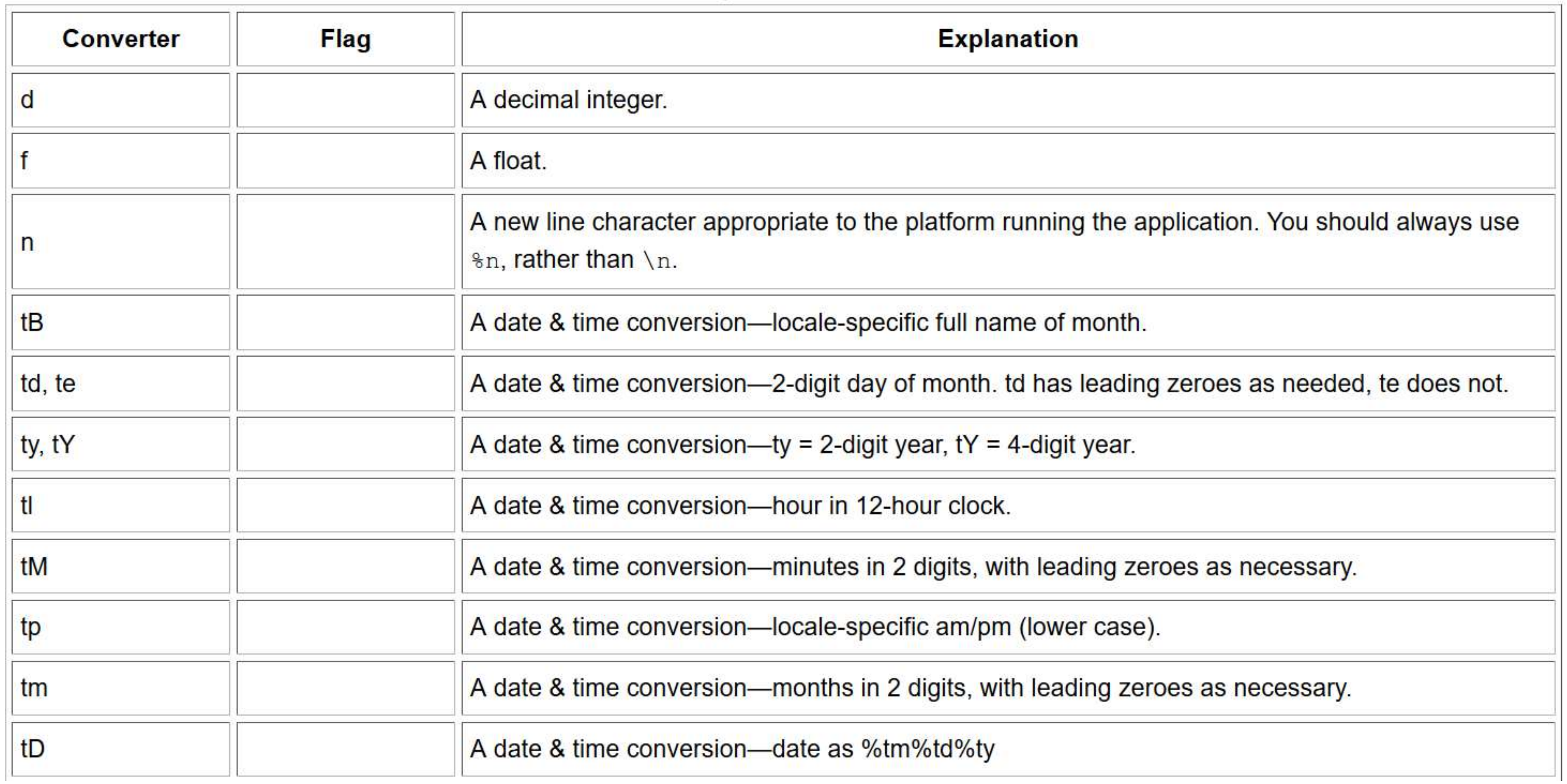
- `StringBuffer` objects are like `String` objects, but they can be changed -> changeable strings
- `StringBuffer` is a thread-safe, changeable string -> methods are synchronised as needed, so that all operations on a specific instance behave as if they occur in a serial order, which matches the order of the method calls of the individual participating threads.
- Most important operations: append() (appends at the end of the buffer) and insert() (inserts the characters at a specific location)
- https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html

Outputting formatted text

- You are already familiar with the use of the print() and println() methods for printing strings as standard output (System.out).
- The java.io package contains a PrintStream class with two formatting methods: format() and printf() (are equivalent)
- System.out == PrintStreamObject
  - System.out.format(.....)
  - System.out.printf(.....)
- Example:

```
float floatVar = 3.45f;
String stringVar = "Auto";
System.out.format("%s: %f Euro\n", stringVar, floatVar);
```

Example: converter for formatted text



Converters and Flags Used in `TestFormat.java`

Converter	Flag	Explanation
d		A decimal integer.
f		A float.
n		A new line character appropriate to the platform running the application. You should always use `%n`, rather than `\n`.
tB		A date & time conversion—locale-specific full name of month.
td, te		A date & time conversion—2-digit day of month. td has leading zeroes as needed, te does not.
ty, tY		A date & time conversion—ty = 2-digit year, tY = 4-digit year.
tl		A date & time conversion—hour in 12-hour clock.
tM		A date & time conversion—minutes in 2 digits, with leading zeroes as necessary.
tp		A date & time conversion—locale-specific am/pm (lower case).
tm		A date & time conversion—months in 2 digits, with leading zeroes as necessary.
tD		A date & time conversion—date as %tm%td%ty

https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Date formatting

```
Calendar c = Calendar.getInstance();

System.out.format("%te. %tB %tY%n", c, c, c);
// --> 30 November 2019

System.out.format("%tl:%tM %tp%n", c, c, c);
// -->  8:54 am

System.out.format("%tD%n", c);
// -->  11/30/19
```

String format

➢ The String class provides the ability to pre-format the string to be created:

```
String fs;
fs = String.format("The value of the float " +
                   "variable is %f, while " +
                   "the value of the " +
                   "integer variable is %d, " +
                   " and the string is %s",
         floatVar, intVar, stringVar);
System.out.println(fs);
```

More on this here:
https://docs.oracle.com/javase/tutorial/essential/io/formatting.html