

## 4. Klassen zur Datenkommunikation

Bei der Erstellung von Anwendungen zum Themenschwerpunkt Datenkommunikation kann auf existierende Klassen zurückgegriffen werden, welche die grundlegenden Methoden bereitstellen. Die Klassen müssten in der jeweiligen Zielsprache implementiert werden.

### Parallele Schnittstellen

Die Kommunikation über parallele Schnittstellen erfolgt über direkten Port-Zugriff. Die Klasse `IOPortAccess` stellt die entsprechenden Methoden zur Ansteuerung von Ports zur Verfügung. (vgl. Klassen-Diagramm)

<b>IOPortAccess</b>
+ IOPortAccess() + openDriver(): boolean + closeDriver(): void + writePort(port: int, value: int): void + readPort(port: int): int

Kurzbeschreibung der Klasse `IOPortAccess`

<code>IOPortAccess()</code>	der Konstruktor
<code>openDriver()</code>	öffnet den Port-Treiber; liefert <code>true</code> , wenn der Treiber geladen wurde
<code>closeDriver()</code>	schließt den Port-Treiber
<code>writePort()</code>	schreibt einen Wert <code>value</code> auf den Port mit der Adresse <code>port</code>
<code>readPort()</code>	liest einen Wert vom Port mit der Adresse <code>port</code>

### Serielle Schnittstellen

Zur Kommunikation über serielle Schnittstellen werden vom Betriebssystem Funktionen bereitgestellt (Win-API). Diese werden in der Klasse `Serial` gekapselt. Die Klasse `Serial` stellt entsprechenden Methoden zur Ansteuerung der seriellen Schnittstelle zur Verfügung. (vgl. Klassen-Diagramm)

<b>Serial</b>
- portName: String - baudrate: int - dataBits: int - stopBits: int - parity: int
+ Serial(String portName, int baudrate, int dataBits, int stopBits, int parity) + open(): boolean + close(): void + dataAvailable(): int + read(): int + read(b: byte[], len: int): int + readLine(): String + write(value: int): void + write(b: byte[], len: int): void + write(s: String): void + setRTS(arg: boolean): void + setDTR(arg: boolean): void + isCTS(): boolean + isDSR(): boolean

### Kurzbeschreibung der Klasse `Serial`

<code>Serial()</code>	der Konstruktor Initialisiert die serielle Schnittstelle ohne sie zu öffnen. portName: Name des Ports, z.B. COM2 baud: Baudrate dataBits: Datenbitanzahl parity: Parität stopBits: Stopbitanzahl
<code>open()</code>	öffnet die serielle Schnittstelle; liefert <code>true</code> , wenn die Schnittstelle verwendbar ist.
<code>close()</code>	schließt die serielle Schnittstelle; die Schnittstelle ist dann solange nicht mehr verwendbar, bis sie wieder geöffnet wird.
<code>dataAvailable()</code>	liefert die Anzahl der Byte, die von der seriellen Schnittstelle gelesen werden können.
<code>read(): int</code>	liest ein Byte (0..255) von der serielle Schnittstelle; Ist kein Byte verfügbar, liefert sie -1.
<code>read(b: byte[], len: int)</code>	liest mehrere Bytes von der seriellen Schnittstelle in ein Byte-Array; liefert die Anzahl der gelesenen Bytes oder -1, wenn keine Bytes verfügbar sind.
<code>readLine(): String</code>	liest eine Zeile von der serielle Schnittstelle; eine Zeile wird durch ein Zeilenendezeichen abgeschlossen; das Zeilenendezeichen wird jedoch nicht in den zurückgegebene String übernommen.
<code>write(value: int)</code>	schreibt ein Byte auf die serielle Schnittstelle; ist die Schnittstelle nicht geöffnet geschieht nichts.
<code>write(b: byte[], len: int)</code>	schreibt mehrere Bytes auf die serielle Schnittstelle; ist die Schnittstelle nicht bereit geschieht nichts.
<code>write(s: String)</code>	schreibt einen String auf die serielle Schnittstelle; ist die Schnittstelle nicht geöffnet geschieht nichts.
<code>setRTS()</code>	setzt den Modem-Steuerausgangs RTS (request to send).
<code>setDTR()</code>	setzt den Modem-Steuerausgangs DTR (data terminal ready).
<code>isCTS()</code>	liefert den Status des Modem-Meldeeingangs CTS (clear to send).
<code>isDSR()</code>	liefert den Status des Modem-Meldeeingangs DSR (data set ready).

### Sockets

Bei der Erstellung der Anwendung zur Kommunikation im Netz kann auf eine existierende Klassen `Socket` und `ServerSocket` zurückgegriffen werden, welche die grundlegenden Methoden zur Kommunikation via TCP/IP bereitstellen. (vgl. Klassen-Diagramm)

Server-Sockets werden durch Objekte der Klasse `ServerSocket` dargestellt

<b>ServerSocket</b>
- localPort: int
+ ServerSocket(localPort: int) + accept(): Socket + close(): void

Kurzbeschreibung der Klasse `ServerSocket`:

<code>ServerSocket()</code>	erzeugt einen Server-Socket, bindet ihn an den angegebenen Port und versetzt ihn in den Horchzustand.
<code>accept()</code>	wartet darauf, dass ein Client eine Verbindung aufbauen will. Wurde eine Verbindung aufgebaut, liefert die Methode das entsprechende Socket-Objekt
<code>close()</code>	schließt den Server-Socket

Sockets werden durch Objekte der Klasse `Socket` dargestellt

<b>Socket</b>
- remoteHostIP: String - remotePort: int
+ Socket(remoteHostIP: String, remotePort: int) + connect(): boolean + dataAvailable(): int + read(): int + read(b: byte[], len: int): int + readLine(): String + write(value: int): void + write(b: byte[], len: int): void + write(s: String): void + close(): void

Kurzbeschreibung der Klasse `Socket`:

<code>Socket()</code>	erzeugt einen Socket
<code>connect()</code>	stellt eine Verbindet zu der bei der Socket-Erzeugung angegebenen IP- Adresse und Port her; liefert <code>true</code> , wenn die eine Verbindung hergestellt werden konnte.
<code>dataAvailable()</code>	liefert die Anzahl der Byte, die vom Socket gelesen werden können.
<code>read(): int</code>	liest ein Byte (0..255) vom Socket; ist kein Byte verfügbar, liefert sie -1.
<code>read(b: byte[], len: int)</code>	liest bis zu <code>len</code> Bytes vom Socket in ein Byte-Array; die Anzahl der tatsächlich gelesenen Bytes wird als zurückgeliefert oder -1, wenn keine Bytes verfügbar sind.
<code>readLine(): String</code>	liest eine Zeile von der serielle Schnittstelle; eine Zeile wird durch ein Zeilenendezeichen abgeschlossen; das Zeilenendezeichen wird jedoch nicht in den zurückgegebene String übernommen.
<code>write(value: int)</code>	schreibt ein Byte (0..255) auf den Socket ; ist keine Verbindung hergestellt, geschieht nichts.
<code>write(b: byte[], len: int)</code>	schreibt <code>len</code> Bytes zum Socket; ist keine Verbindung hergestellt, geschieht nichts.
<code>write(s: String)</code>	schreibt einen String zum Socket; ist die Schnittstelle nicht geöffnet geschieht nichts.
<code>close()</code>	löst die Verbindung auf

Hinweis: Bei der Realisierung der Socket-Klasse wurden bewusst Methoden implementiert, die analog zur seriellen Schnittstelle verwendet werden können.