

Hinweise für den Prüfling

Bearbeitungszeit: 300 Minuten

Auswahlverfahren

Wählen Sie von den zwei vorliegenden Vorschlägen einen zur Bearbeitung aus. Der nicht ausgewählte Vorschlag wird 60 Minuten nach Beginn der Bearbeitungszeit von der Aufsicht führenden Lehrkraft eingesammelt.

Erlaubte Hilfsmittel

1. ein Wörterbuch der deutschen Rechtschreibung
2. ein eingeführter Taschenrechner (Bei grafikfähigen Rechnern und ComputeralgebraSystemen ist ein Reset durchzuführen.)
3. eine Liste der fachspezifischen Operatoren Fachbereich III

Sonstige Hinweise

keine

In jedem Fall vom Prüfling auszufüllen

Name: _____

Vorname: _____

Prüferin/Prüfer: _____

Datum: _____

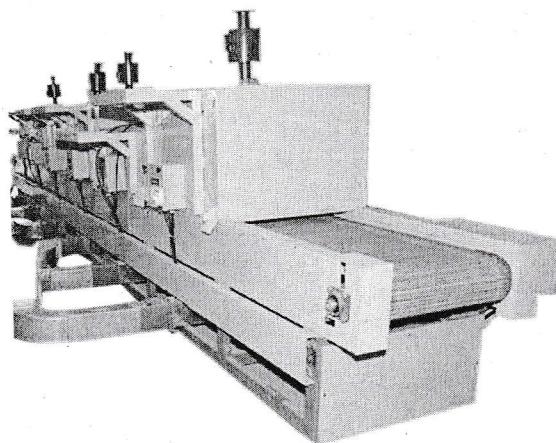
Industrielle Anlage zum Aushärten von Elektronikbauteilen

Aufgaben

Ein Industrieunternehmen produziert verschiedene Elektronikbauteile, z.B. Platinen für die Flugzeug- und die Autoindustrie. Der letzte Produktionsschritt ist das Aushärten der Teile. Die hierzu verwendete industrielle Anlage besteht aus einem Förderband und einem Brennofen. Die Elektronikbauteile werden auf das Förderband gelegt. Wenn der Brennofen die richtige Temperatur erreicht hat, wird das Band gestartet und die Teile durch den Brennofen transportiert. Die Geschwindigkeit des Bandes ergibt sich aus der notwendigen Brenndauer. Nach dem Härtvorgang werden die Teile vom Band entnommen.

Neben einer Heizung mit Temperatursensor ist der Brennofen mit einem Energy-Meter ausgestattet, um den Energieverbrauch zu messen. Zum Steuern der industriellen Anlage soll eine objektorientierte Anwendung entwickelt werden. Die Komponenten der Anlage sollen über einen seriellen Controller angesprochen werden. Die Beschreibung des Kommunikationsprotokolls finden Sie in Material 1.

Hinweis: Das Auflegen der Bauteile auf das Förderband sowie das Entnehmen der Bauteile vom Band sind nicht Gegenstand dieser Aufgabe.



<https://4.imimg.com/data4/AJ/JE/MY-948577/belt-conveyor-oven-500x500.jpg> (abgerufen am 26.04.2020).

1 Entwicklung einer Anlagensteuerung für die industrielle Anlage

1.1 Sie finden ein erstes UML-Klassendiagramm der Anlagensteuerung in Material 2.

1.1.1 Die Klasse **Komponente** ist als abstrakte Klasse modelliert. Beschreiben Sie das Konzept und den Nutzen abstrakter Klassen in der objektorientierten Programmierung. **(2 BE)**

1.1.2 Implementieren Sie die Klassen **Komponente** und **Heizung** in einer objektorientierten Programmiersprache.

Hinweise: Byte-Operationen sollen mithilfe der Klasse **ByteBuffer** (Material 3) durchgeführt werden. Die Methoden der Komponenten (Band, Heizung, Energy-Meter) erzeugen Byte-Arrays (Länge 7) für die Kommunikation zwischen der Klasse **Anlage** und dem Controller der realen Anlage. Die Prüfsumme wird nicht in diesen Methoden, sondern erst später, eingefügt.

(10 BE)

III Bewertung und Beurteilung

Die Bewertung und Beurteilung erfolgt unter Beachtung der nachfolgenden Vorgaben nach § 33 der Oberstufen- und Abiturverordnung (OAVO) in der jeweils geltenden Fassung. Bei der Bewertung und Beurteilung der sprachlichen Richtigkeit in der deutschen Sprache sind die Bestimmungen des § 9 Abs. 12 Satz 3 OAVO in Verbindung mit Anlage 9b anzuwenden.

In den modernen Fremdsprachen ist nach den Bestimmungen des § 9 Abs. 13 OAVO in Verbindung mit dem „Erlass zur Bewertung und Beurteilung von schriftlichen Arbeiten in allen Grund- und Leistungskursen der neu beginnenden und fortgeführten modernen Fremdsprachen in der gymnasialen Oberstufe, dem beruflichen Gymnasium, dem Abendgymnasium und dem Hessenkolleg“ vom 7. August 2020 (ABl. S. 519) die sprachliche Leistung kriteriengeleitet zu bewerten. Bei der Bewertung und Beurteilung der Übersetzungsleistung in den Fächern Latein und Altgriechisch sind die Bestimmungen des § 9 Abs. 14 OAVO in Verbindung mit Anlage 9c anzuwenden.

Der Fehlerindex ist nach Anlage 9b zu § 9 Abs. 12 OAVO zu berechnen. Für die Ermittlung der Punkte nach Anlage 9a zu § 9 Abs. 12 OAVO sowie Anlage 9c zu § 9 Abs. 14 OAVO wird jeweils der ganzzahlige nicht gerundete Prozentsatz bzw. Fehlerindex zugrunde gelegt.

Darüber hinaus sind die Vorgaben der Erlasse „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen (Abiturerlass)“, „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen im beruflichen Gymnasium (fachrichtungs-/schwerpunktbezogene Fächer) (Abiturerlass BG)“ und „Durchführungsbestimmungen zum Landesabitur“ in der für den Abiturjahrgang geltenden Fassung zu beachten.

Bei der Bewertung und Beurteilung ist auch die Intensität der Bearbeitung zu berücksichtigen. Als Bewertungskriterien dienen über das Inhaltliche hinaus qualitative Merkmale wie Strukturierung, Differenziertheit, sprachliche Gestaltung und Schlüssigkeit der Argumentation.

Im Fach Praktische Informatik besteht die Prüfungsleistung aus der Bearbeitung eines Vorschlags, wofür insgesamt maximal 100 BE vergeben werden können. Ein Prüfungsergebnis von **5 Punkten** (ausreichend) setzt voraus, dass mindestens 45% der zu vergebenden BE erreicht werden. Ein Prüfungsergebnis von **11 Punkten (gut)** setzt voraus, dass mindestens 75% der zu vergebenden BE erreicht werden.

Gewichtung der Aufgaben und Zuordnung der Bewertungseinheiten zu den Anforderungsbereichen

Aufgabe	Bewertungseinheiten in den Anforderungsbereichen			Summe
	AFB I	AFB II	AFB III	
1	15	27	18	60
2	12	18	10	40
Summe	27	45	28	100

Die auf die Anforderungsbereiche verteilten Bewertungseinheiten innerhalb der Aufgaben sind als Richtwerte zu verstehen.

1.1.3 Implementieren Sie die Methode `setzeTemperatur()` der Klasse `Anlage`. Sie liefert 0 als Rückgabewert, wenn die Temperatur gesetzt werden konnte, -1 wenn eine zu hohe Temperatur übergeben wurde, andernfalls den Fehlercode des Anlagencontrollers.

Hinweise: Nach dem Senden der Nachricht (Byte-Array mit Prüfsumme) muss 100 Millisekunden gewartet werden, bevor die Antwort der Anlage gelesen werden kann. Der Controller der Anlage antwortet jeweils mit der identischen Nachricht, falls er einen Befehl ausführen konnte. Schlägt die Befehlausführung fehl, steht im Datenwert der Nachricht ein 4 Byte-Fehlercode im Integerformat.

(9 BE)

1.1.4 Für jede Nachricht an den Controller der realen Anlage wird eine Prüfsumme von 1 Byte benötigt. Sie wird nach dem Algorithmus des Struktogramms in Material 4 berechnet.
Implementieren Sie die Methode `berechneChecksum()` der Klasse `Anlage`.

(6 BE)

1.1.5 Die Methode `starteProzess()` der Klasse `Steuerung` steuert den Brennprozess für ein bestimmtes Bauteil nach folgendem Ablauf:

- schalte Heizung ein,
- setze Temperatur der Heizung für das Bauteil,
- warte bis Heizung gewünschte Temperatur erreicht hat,
- setze Bandgeschwindigkeit für das Bauteil,
- schalte Band ein.

Zeichnen und modellieren Sie ein UML-Sequenzdiagramm für die Methode. Eine Vorlage finden Sie in Material 5.

Hinweis: Eine Fehlerbehandlung muss nicht modelliert werden.

(10 BE)

1.2 Für die Steuerung der Brennvorgänge wird ein Leitstand-System (User-Interface) entwickelt.

1.2.1 Beschreiben Sie das in Material 6 dargestellte unvollständige UML-Anwendungsfalldiagramm sowie die Bedeutung der Komponenten.

(4 BE)

1.2.2 Entwickeln und zeichnen Sie für das Leitstand-System aufgrund folgender Anforderungen ein Anwendungsfalldiagramm in UML-Notation:

- Ein Produktionsplaner oder eine -planerin plant den Brennvorgang eines Bauteils.
- Ein Mitarbeiter oder eine Mitarbeiterin in der Produktion (im Folgenden Mitarbeiter genannt) startet die Anlage.
- Der Mitarbeiter fragt den Zustand der Anlage ab. Falls eine Störung auftritt, muss die Anlage per Notabschaltung angehalten werden.
- Wird der Brennvorgang normal beendet, wird die Anlage durch den Mitarbeiter abgeschaltet.
- Der Schichtleiter oder die Schichtleiterin kann bei einem Personalausfall alle Aufgaben im System übernehmen.

Hinweis: Das UML-Anwendungsfalldiagramm in Material 6 ist zu ergänzen.

(6 BE)

- 1.3 Der Energieverbrauch der Brennvorgänge soll überwacht werden. Dafür kennt die Steuerung ein Exemplar der Klasse `DataLogger`. Die Klasse `DataLogger` verwaltet die Daten der Brennvorgänge in einer Liste.

Für jeden Brennvorgang werden folgende Daten gespeichert:

- `start` (die Startzeit als `DateTime`),
- `ende` (Endzeitpunkt als `DateTime`),
- produziertes Bauteil,
- Energieverbrauch in kWh.

Einen Entwurf des erweiterten Klassendiagramms finden Sie in Material 7. Die Methode `startLog()` der Klasse `DataLogger` erzeugt eine Instanz der Klasse `Brenndaten` und protokolliert den Startzeitpunkt und den Zählerstand zu Beginn des Brennvorgangs. Die Methode `endLog()` ermittelt den Endzeitpunkt des Brennvorgangs, den dann aktuellen Zählerstand und damit den Energieverbrauch.

Die Klasse `DataLogger` hat eine Methode `ausgabeTagesuebersicht()` zur Auswertung der Energiedaten. Pro Tag werden mehrere Brennvorgänge durchgeführt, die Methode gibt eine Übersicht nach folgendem Format auf der Konsole aus:

Energieübersicht pro Tag

12.03.2021	78,34 kWh
13.03.2021	71,96 kWh
14.03.2021	69,77 kWh

...

Die Methode `leseEMeterzaehler()` der Klasse `Anlage` (Material 2) liefert den Zählerstand zur Ermittlung des Energieverbrauchs.

- 1.3.1 Modellieren Sie die notwendigen Assoziationen, Multiplizitäten und Rollen unter Verwendung der Vorlage in Material 7. (3 BE)

- 1.3.2 Überführen Sie die Klasse `DataLogger` in entsprechende Anweisungen einer objektorientierten Programmiersprache und implementieren Sie die aufgeführten Methoden. (10 BE)

- 2 Datenbank für den Produktionsbetrieb
Das Unternehmen setzt mehrere Maschinen ein, die regelmäßig gewartet werden müssen. Sowohl die Daten der Maschinen als auch die Bestellungen von Kunden sollen in einer relationalen Datenbank gespeichert werden. Ein erstes Datenmodell finden Sie in Material 8. Das Datenmodell soll später erweitert werden.

- 2.1 Überführen Sie das gegebene ER-Modell in das relationale Modell unter Einhaltung der 3. Normalform und begründen Sie Ihre Entscheidungen.
Hinweis: Alle Relationen sind in der Schreibweise `Relation (PK, Attribut, ..., FK#)` anzugeben. (9 BE)

- 2.2 Die Datenbank soll um eine Tabelle ergänzt und Daten sollen geändert bzw. ausgewertet werden.
- 2.2.1 Formulieren Sie eine SQL-Anweisung, die die Tabelle `Produktionslauf` in der Datenbank anlegt. (3 BE)
- 2.2.2 Geben Sie eine SQL-Anweisung an, die die Brenndauer der Bauteile, die noch nicht produziert wurden, um 10% erhöht. (3 BE)
- 2.2.3 Heinz Müller (Telefon 0152/7766889) ist neuer Mitarbeiter und soll die Verantwortung für alle Maschinen erhalten, die zurzeit keinen Verantwortlichen haben. Implementieren Sie die SQL-Anweisungen zur Aktualisierung der Datenbank.
Hinweis: Die Spalte `personalnr` in der Tabelle `Mitarbeiter` ist mit der Option `auto_increment` deklariert. (4 BE)
- 2.2.4 Entwickeln Sie eine SQL-Anweisung, die eine Liste aller Maschinentypen mit Wartungsintervall und Anzahl der Maschinen, absteigend sortiert nach Anzahl der Maschinen, ausgibt. (4 BE)
- 2.2.5 Entwickeln Sie eine SQL-Anweisung, die eine Liste aller Maschinen mit ihrer Laufzeit seit der letzten Wartung und dem Wartungsintervall erstellt, die sofort gewartet werden müssen. (5 BE)
- 2.3 Das Unternehmen möchte neue Anforderungen im Datenmodell umsetzen. Dazu muss das ER-Modell aus Material 8 geändert und erweitert werden:
- Die Kunden des Unternehmens, die mit Firmennamen und Adresse zu speichern sind, geben Bestellungen auf. Das Datum der Bestellung ist festzuhalten.
 - Bestellungen bestehen aus mehreren Bestellpositionen, die sich auf genau ein Produkt des Unternehmens beziehen. Für jede Bestellposition sind die Positionsnummer, die bestellte Menge und der Verkaufspreis festzuhalten.
 - Ein Produkt besteht aus mehreren Bauteilen in unterschiedlicher Anzahl. Jedes Bauteil kann in verschiedenen Produkten zum Einsatz kommen.
- 2.3.1 Entwickeln und zeichnen Sie das erweiterte ER-Modell in [min, max]-Notation. (9 BE)
- 2.3.2 Das Unternehmen entscheidet sich, auch externe Bauteile im Rahmen der Produktion zu verwenden. Dazu wurde ein erweitertes ER-Modell entwickelt (siehe Material 9). Überführen Sie das erweiterte ER-Modell in das relationale Modell und nennen Sie jeweils einen Vor- und Nachteil der von Ihnen gewählten Lösung. (3 BE)

Material 1**Kommunikationsprotokoll für den Brennofen**

Der Brennofen ist mit Sensoren und Aktoren (allgemein: Komponenten) ausgestattet, die von einem Controller gesteuert werden. Der Controller ist mit einem Steuer-PC über die serielle Schnittstelle verbunden. Er nimmt Nachrichten über die serielle Schnittstelle vom PC entgegen und kommuniziert intern mit seinen Komponenten.

Kommunikationsprotokoll

Jede Nachricht des PC besteht aus 7 Bytes, die wie folgt aufgebaut ist:

Sensor/Aktor (Kennung)	Function-Code	Datenwert	Checksum
1 Byte	1 Byte	4 Byte	1 Byte

Beispiel: Wenn ein Bauteil mit 140 Grad Celsius und einer Bandgeschwindigkeit von 4 Metern pro Minute die Anlage passieren soll, sendet die Steuerung folgende Nachrichten über die serielle Schnittstelle.

Nachrichtensequenz für das Beispiel

Sensor / Aktor (Kennung)	Funktionscode	4 Datenbytes				Prüfsumme (Checksum)	Bedeutung
54 = Heizung, 42 = Band, 45 = Energy-Meter	03 = Wert lesen, 05 = schalten, 06 = Wert schreiben	00	00	00	0F	84	Heizung einschalten
54	05	00	00	00	8C	27	Temperatur auf 140° C setzen
54	06	00	00	00	00	CC	Temperatur abfragen
54	03	00	00	00	8C	6F	Antwort der Anlage: 140° C
54	03	00	00	00	04	D5	Bandgeschwindigkeit auf 4 m/min setzen
42	06	00	00	00	04	70	Band einschalten
42	05	00	00	00	0F	2C	Heizung ausschalten
54	05	00	00	00	F0	D8	Band ausschalten

Der Brennofen antwortet jeweils mit der identischen Nachricht, falls der Befehl ausgeführt werden konnte. Die identische Nachricht ist in der Tabelle nicht dargestellt. Schlägt die Befehlsausführung fehl, steht im Datenwert ein 4-Byte-Fehlercode. Bei Verwendung des Funktionscodes **schalten** kennzeichnet der Wert **0F** im Datenbyte 4 die Funktion **einschalten**, der Wert **F0** die Funktion **ausschalten**.

Material 3**Klassendokumentationen****Klasse ByteBuffer**

Ein ByteBuffer ist einem Byte-Array sehr ähnlich; seine maximale Größe wird als erstes mit der Methode `allocate()` festgelegt und kann später nicht dynamisch wachsen. Ist ein ByteBuffer angelegt, so können über einen Index bei 0 beginnend die einzelnen Bytes gelesen und geschrieben werden. Die aktuelle Position innerhalb des ByteBuffers ist der Indexwert, von dem als nächstes gelesen bzw. zu dem als nächstes geschrieben wird. Im Gegensatz zu einem Array vom Typ `byte` besitzt die Klasse `ByteBuffer` Methoden, die die einfache Manipulation der Inhalte gestatten, wie z.B. `putInt()` und `getInt()`.

ByteBuffer
+ <code>allocate(n : int) : ByteBuffer</code>
+ <code>wrap(array : byte[]) : ByteBuffer</code>
+ <code>put(b : byte)</code>
+ <code>put(index : int, b : byte)</code>
+ <code>put(src : byte[], offset : int, length : int)</code>
+ <code>putInt(value : int)</code>
+ <code>rewind()</code>
+ <code>array() : byte[]</code>
+ <code>equals(bb : ByteBuffer) : boolean</code>
+ <code>getInt() : int</code>

`allocate(n: int) : ByteBuffer`
 legt einen ByteBuffer mit n Bytes an, der mit Nullen initialisiert wird.

`wrap(array: byte[]): ByteBuffer`
 konvertiert das Byte-Array array in ein ByteBuffer-Objekt.

`put(b: byte)`
 schreibt Byte b in den Puffer an die aktuelle Position.

`put(index: int, b: byte)`
 schreibt Byte b an Position index im Puffer.

`put(src: byte[], offset: int, length: int)`
 schreibt length Bytes des Arrays src in den Puffer beginnend an der Position offset des Arrays.

`putInt(value: int)`
 schreibt den int-Wert value mit 4 Bytes in den Puffer an die aktuelle Position.

`rewind()`
 setzt die Schreib-/Leseposition des Puffers auf die Position 0 zurück.

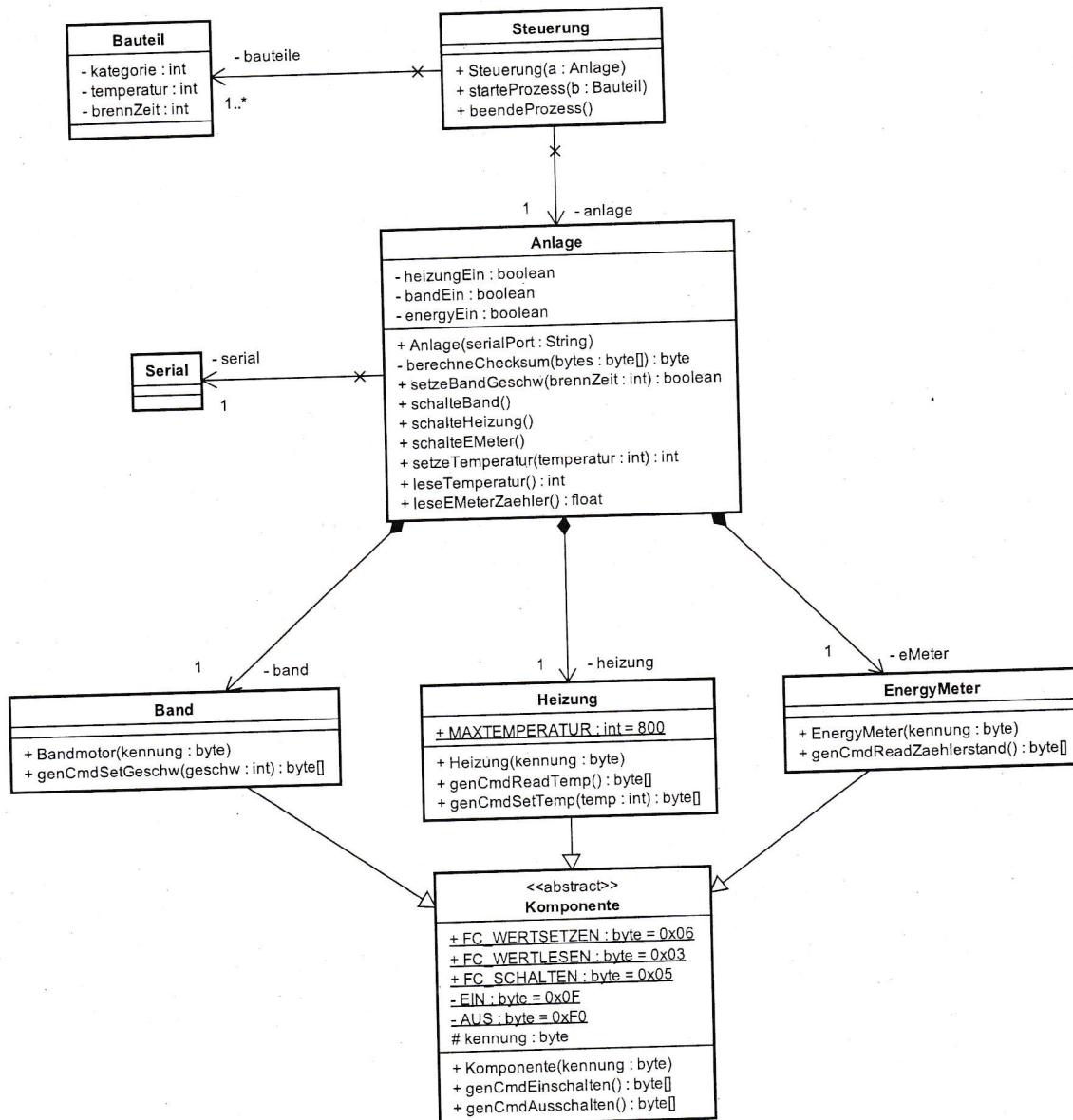
`array(): byte[]`
 konvertiert den Puffer in ein Byte-Array.

`equals(bb: ByteBuffer) : boolean`
 überprüft, ob die Inhalte von zwei ByteBuffer-Objekten identisch sind.

`getInt() : int`
 wandelt 4 Bytes des Puffers in einen Integerwert um, beginnend mit der aktuellen Position.

Material 2

UML-Klassendiagramm



Hinweise: Die Klasse **Anlage** sendet Kommandos über die serielle Schnittstelle an den Controller des Brennofens. Jedes Kommando besteht aus 7 Byte (siehe Material 1). Die Methoden, die mit `genCmd` beginnen, generieren jeweils ein Kommando für die reale Anlage. Jede Komponente der Anlage kann ein- bzw. ausgeschaltet werden. Ebenso können Daten aus Komponenten ausgelesen werden. Die Geschwindigkeit des Bandes wird in Meter pro Minute angegeben, die Temperatur in Grad Celsius. Auf alle Attribute kann mittels get-Methoden zugegriffen werden.

Material 3 (Fortsetzung)**Klasse DateTime**`DateTime()`

erzeugt ein DateTime-Objekt mit dem aktuellen Systemdatum und der Systemuhrzeit, im Format dd.mm.yyyy hh:mm.

`isEqual(dt: DateTime): boolean`
liefert true, wenn das DateTime-Objekt gleich dem des Parameters dt ist.`isBefore(date: DateTime): boolean`
liefert true, wenn das DateTime-Objekt vor dem des Parameters dt liegt.`isAfter(date: DateTime): boolean`
liefert true, wenn das DateTime-Objekt nach dem des Parameters dt liegt`compareTo(dt: DateTime): int`
Vergleicht Datum und Uhrzeit des DateTime-Objekts mit dem des Parameters dt.

Liefert einen int-Wert
< 0 wenn das Datum des DateTime-Objekts vor dem des Parameters dt liegt
= 0 wenn das Datum des DateTime Objekts gleich dem von dt ist
> 0 wenn das Datum des DateTime-Objekts nach dem des Parameters dt liegt

`plusMinutes(minutes: long): DateTime`
liefert eine Kopie des DateTime-Objekts, zu dem die Anzahl Minuten minutes hinzugefügt wurde.`toString(): String`
liefert eine String-Vertretung des DateTime-Objekts im Format dd.mm.yyyy;hh:mm.`toDateString(): String`
liefert eine String-Vertretung des DateTime-Objekts im Format dd.mm.yyyy.

DateTime
+ <code>DateTime()</code>
+ <code>isEqual(dt : DateTime) : boolean</code>
+ <code>isBefore(dt : DateTime) : boolean</code>
+ <code>isAfter(dt : DateTime) : boolean</code>
+ <code>compareTo(dt: DateTime): int</code>
+ <code>plusMinutes(minutes : long) : DateTime</code>
+ <code>toString() : String</code>
+ <code>toDateString() : String</code>

Klasse String`charAt(index: int): char`
liefert das Zeichen an der Stelle index.`compareTo(str: String): int`
vergleicht zwei Strings und liefert einen positiven Rückgabewert, wenn der String lexikografisch größer als str ist und einen negativen, wenn der String kleiner als str ist. Bei identischen Strings ist der Rückgabewert 0.`equals(str: String): boolean`
liefert true, wenn beide Strings gleich sind, andernfalls false.`split(str: String): String[]`
teilt einen String am Trennzeichen str. Die Teil-Strings werden in einem Feld zurückgeliefert.`startsWith(str: String): boolean`
prüft, ob der String mit dem str beginnt.`substring(beginIndex: int, endIndex: int): String`
liefert den Teil-String ab der Stelle beginIndex bis zur Stelle endIndex.

String
+ <code>charAt(index : int) : char</code>
+ <code>compareTo(str : String) : int</code>
+ <code>equals(str : String) : boolean</code>
+ <code>split(str : String) : String[]</code>
+ <code>startsWith(str : String) : boolean</code>
+ <code>substring(beginIndex : int, endIndex : int) : String</code>

Klasse Serial

Ein Exemplar der Klasse `Serial` ermöglicht die Kommunikation über die serielle Schnittstelle.

Kurzbeschreibung der Klasse Serial

`Serial(...)`

Der Konstruktor initialisiert die serielle Schnittstelle, ohne sie zu öffnen.

`portName:` Name des Ports

`baud:` Baudrate

`dataBits:` Datenbitanzahl

`stopBits:` Stopbitanzahl

`parity:` Parität

`open(): boolean`

öffnet die serielle Schnittstelle; liefert `true`, wenn die Schnittstelle verwendbar ist.

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

`close():`

schließt die serielle Schnittstelle; die Schnittstelle ist dann solange nicht mehr verwendbar, bis sie wieder geöffnet wird.

`dataAvailable(): int`

liefert die Anzahl der Bytes, die von der seriellen Schnittstelle gelesen werden können.

`read(): byte`

liest ein Byte (0..255) von der seriellen Schnittstelle, bzw. -1, wenn die Schnittstelle nicht geöffnet ist.

Die Methode blockiert, bis ein Byte verfügbar ist.

`read(b: byte[], len: int)`

liest mehrere Bytes von der seriellen Schnittstelle in ein Byte-Array; liefert die Anzahl der gelesenen Bytes, bzw. -1, wenn der Schnittstelle nicht geöffnet ist.. Die Methode blockiert, bis mindestens ein Byte verfügbar ist.

`readLine(): String`

liest eine Zeile von der seriellen Schnittstelle, bzw. liefert null, wenn die Schnittstelle nicht geöffnet ist. Eine Zeile wird durch ein Zeilenendezeichen abgeschlossen; das Zeilenendezeichen wird jedoch nicht in den zurückgegebenen String übernommen. Die Methode blockiert, bis eine komplette Zeile eingelesen ist.

`write(value: int)`

schreibt ein Zeichen auf die serielle Schnittstelle; ist die Schnittstelle nicht geöffnet geschieht nichts.

`write(b: byte[], len: int)`

schreibt mehrere Bytes auf die serielle Schnittstelle; ist die Schnittstelle nicht bereit geschieht nichts.

`write(s: String)`

schreibt einen String auf die serielle Schnittstelle; ist die Schnittstelle nicht geöffnet geschieht nichts.

Material 4**Struktogramm zum Berechnen der Prüfsumme****berechneCheckSum(byte[] bytes)**

crc als byte

crc := 0xFF

zähle pos von 0 bis 5, Schrittweite 1

crc := crc XOR bytes[pos]

zähle i von 1 bis 8, Schrittweite 1

Bit 0 gesetzt?

T

F

schiebe Bits in crc um 1 nach rechts

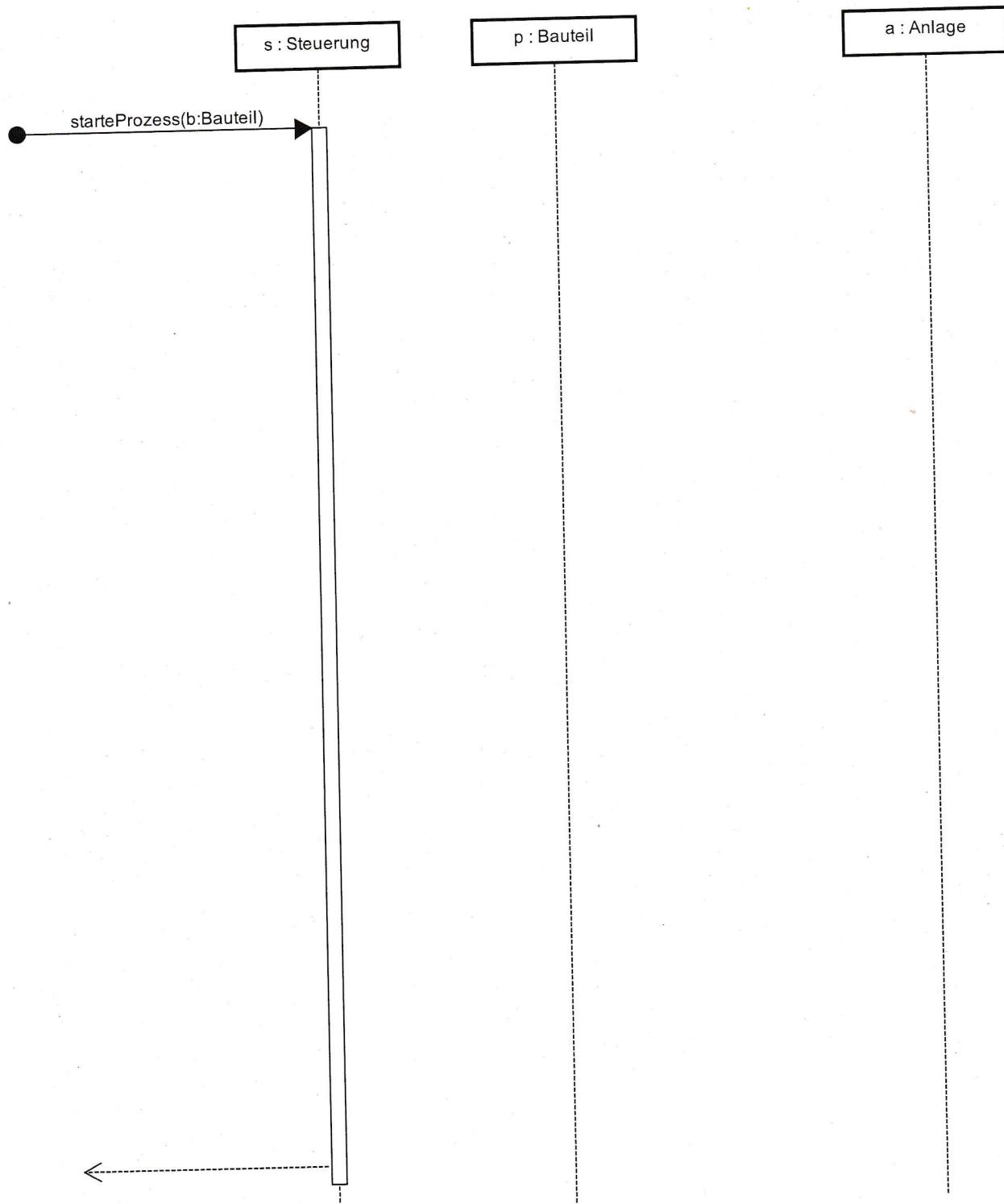
schiebe Bits in crc um 1 nach rechts

crc := crc XOR 0xA1

return crc

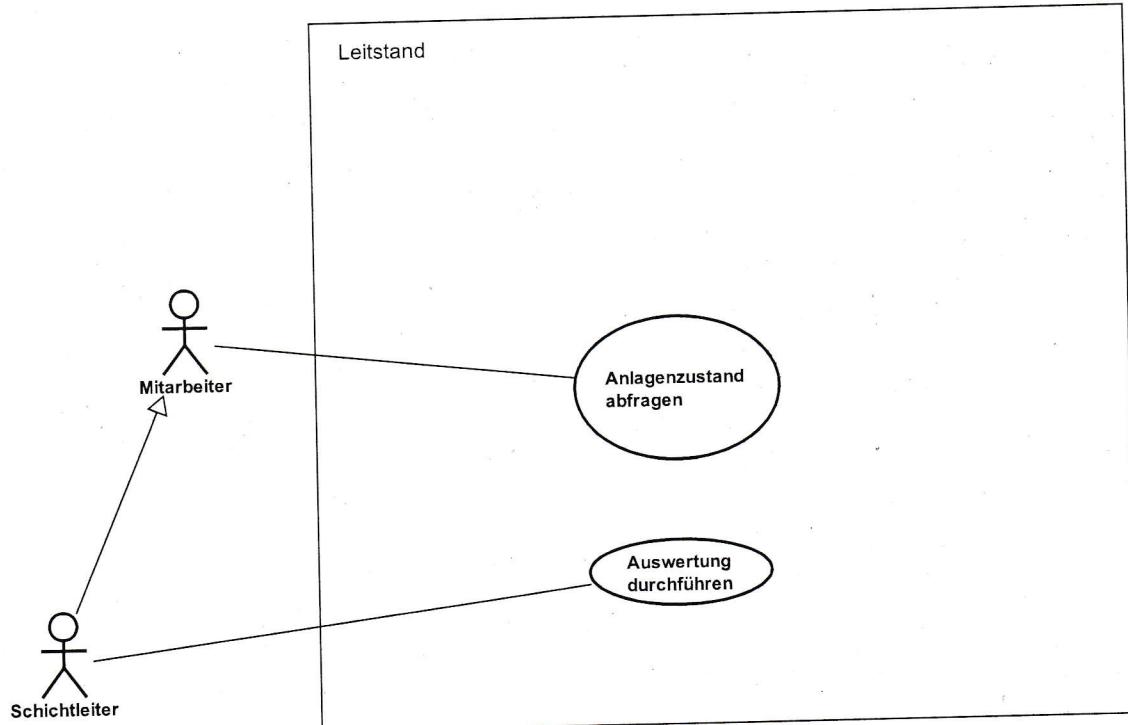
Material 5

Vorlage UML-Sequenzdiagramm



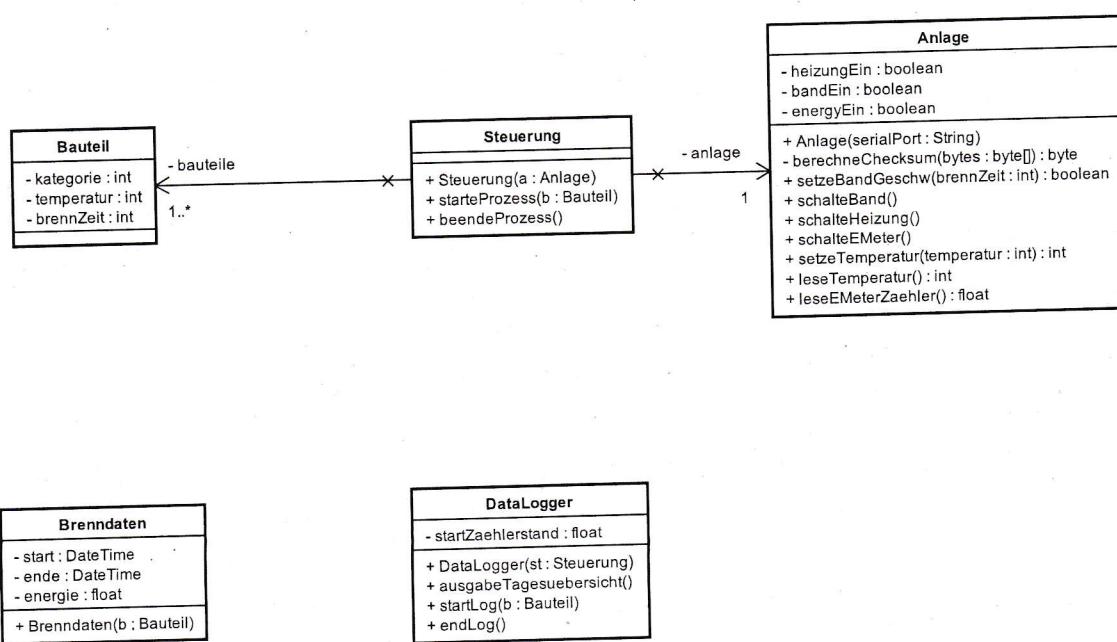
Material 6

UML-Anwendungsfalldiagramm



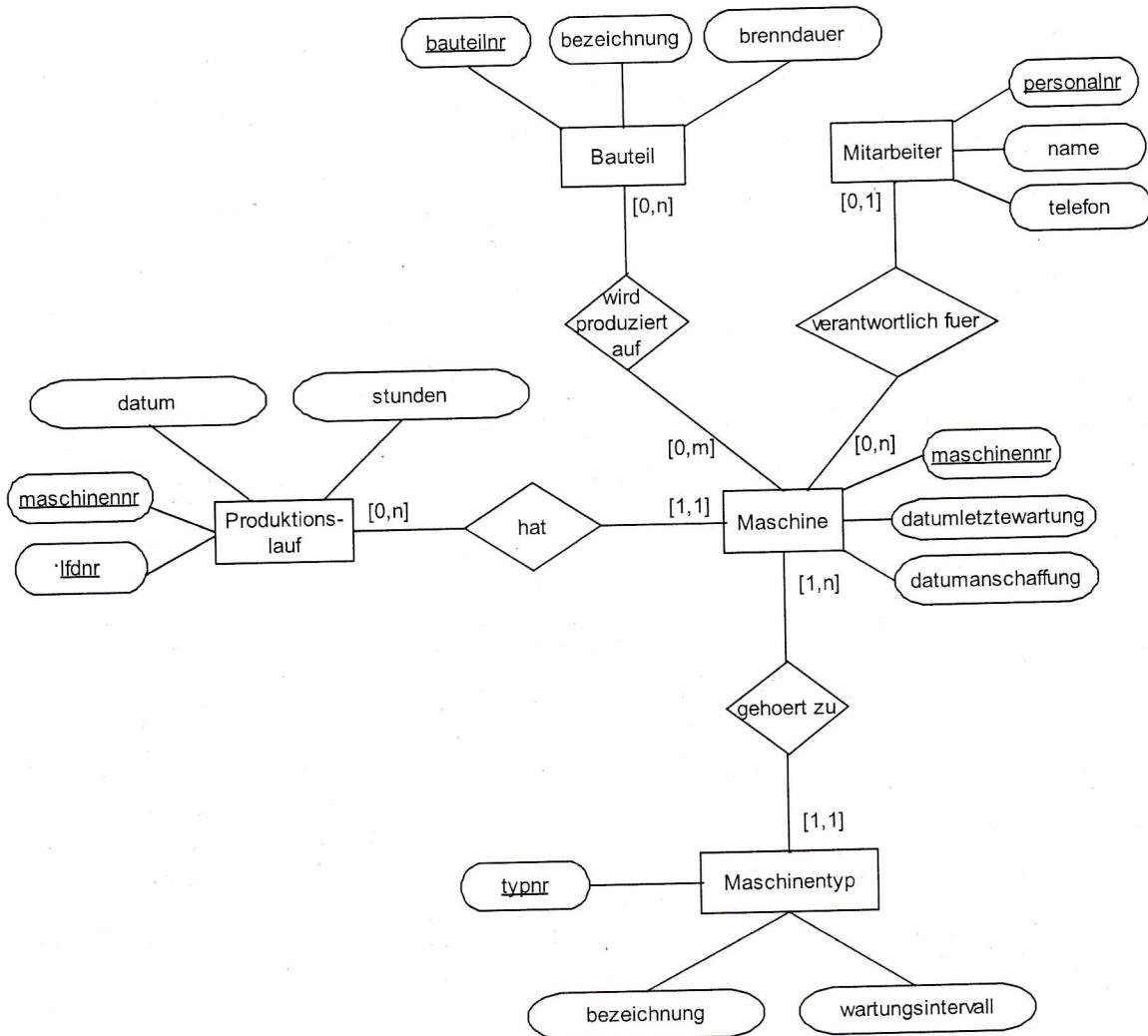
Material 7

Vorlage UML-Klassendiagramm



Material 8

Entity-Relationship-Diagramm



Hinweis: Das Attribut `wartungsintervall` des Entitätstyps `Maschinentyp` ist in Stunden angegeben.

Material 9

Erweitertes Entity-Relationship-Diagramm

