Beispiel 1 - Befehlsinterpreter (55 Punkte)

Ihre Aufgabe besteht darin Teile eines Befehlsinterpreters zu implementieren. Dieser ermöglicht es Befehle auszuführen. Ein Befehl besteht aus einem oder mehreren Kommandos.

Ein Kommando legt eine Aktion auf einer Hardwarekomponente fest. Anhand der in dem Kommando enthaltenen Informationen wird eine Hardwarekomponente ausgewählt und anschließend die durch das Kommando spezifizierte Aktion durchgeführt. Ein Beispiel für eine Aktion ist das Lesen des aktuellen Werts einer ADC Hardwarekomponente (z.B. [ADC, 2, READ]).

Nach der Durchführung eines Kommandos wird mittels eines *Brokers* eine Nachricht an alle Beobachter gesendet. Diese enthält Informationen zu dem durchgeführten Kommando (z.B. [SUCCESS, ADC 2 READ, 135]).

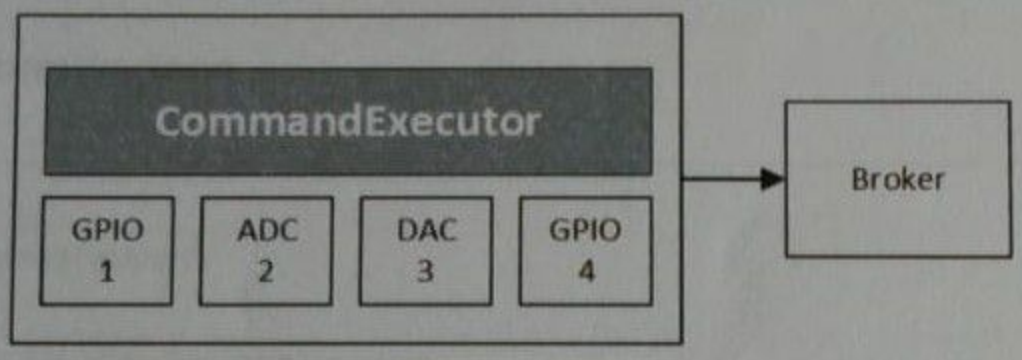

Abbildung 1 Überblick

Jeder Befehl besteht aus einem oder mehreren Kommandos (siehe Abbildung 2). Die Befehle werden durch den Identifier des ersten Kommandos eindeutig identifiziert (z.B. Befehle: D1, A1, A2).

Die Befehle werden aus einer Datei geladen und können danach abgearbeitet werden. Die Abarbeitung kann entweder bei dem ersten Befehl begonnen, oder ab einem bestimmten Befehl fortgesetzt werden.

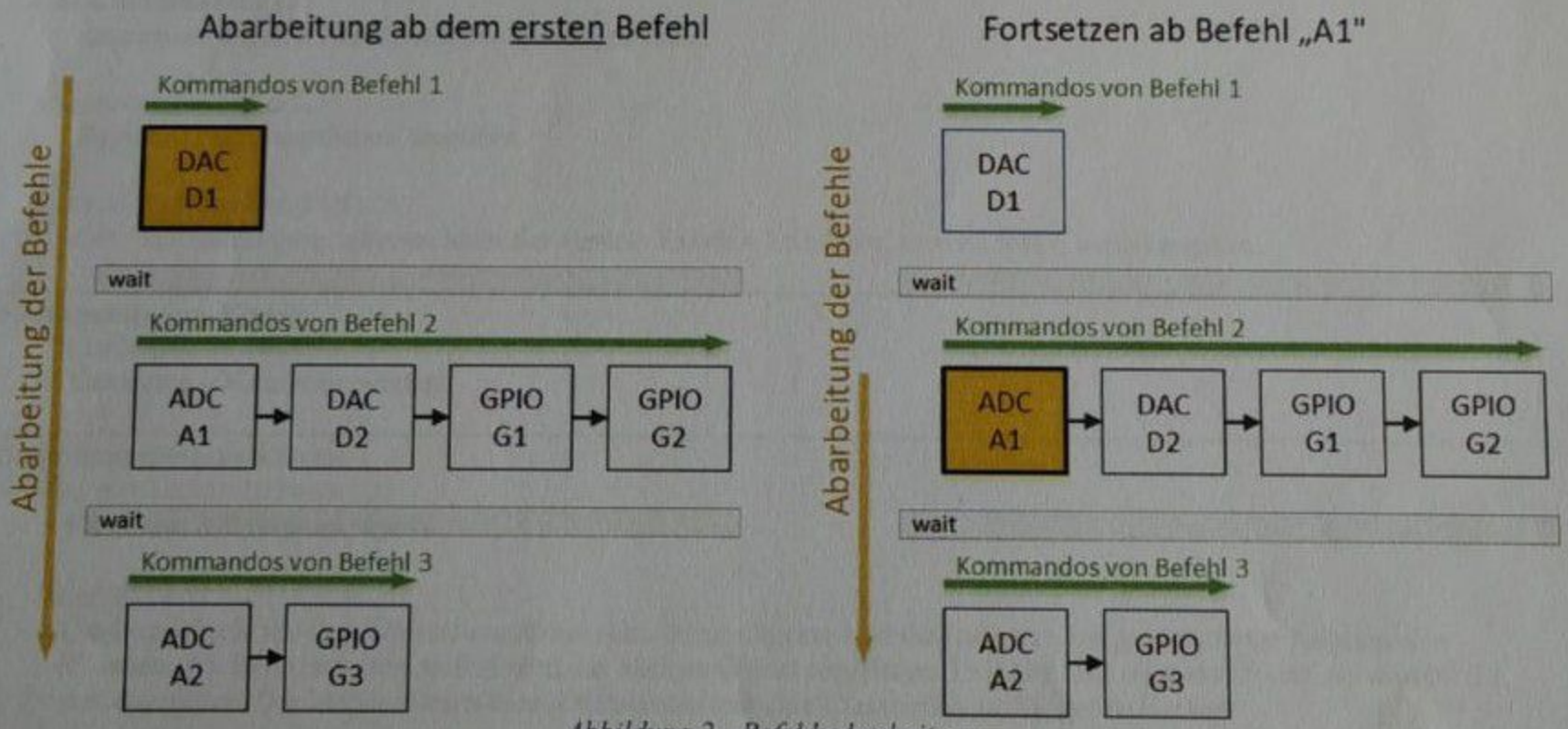

Abbildung 2 - Befehlsabarbeitung

Implementierung

Die im Klassendiagramm rot markierten Klassen sind Klassen, von denen Sie Teile implementieren müssen. Die dazugehörigen Beschreibungen der Methoden finden Sie im Abschnitt *Methoden-Beschreibungen*. Weiters existieren zu diesen Methoden-Beschreibungen Sequenzdiagramme. Bitte beachten Sie, dass die Sequenzdiagramme nur Ausschnitte aus der gesamten Funktionalität der Methode darstellen. Die vollständige Funktionalität ist aus der Beschreibung der Methode zu entnehmen.

Implementieren Sie

- die **komplette `Broker`-Klasse**,
- **die rot markierten Methoden der `CommandExecutor` Klasse**.

Methoden-Beschreibungen

CommandExecutor:

Der Header der Klasse ist nicht gefragt!

`runCommands(...)`
Entnehmen Sie den Ablauf den Sequenzdiagrammen.

`findHardware(...)`
Entnehmen Sie den Ablauf den Sequenzdiagrammen.

`executeOnHardware(...)`
Entnehmen Sie den Ablauf den Sequenzdiagrammen.

`loadCommands(...)`
Entnehmen Sie den Ablauf den Sequenzdiagrammen.

`saveCommandId(...)`
Speichert den übergebenen Identifier.

`loadLastCommandId(...)`
Gibt den zuletzt gespeicherten Identifier zurück. Existiert kein Wert, so wird `NULL` zurückgegeben.

FileInputStream-Klasse:

`FileInputStream(...)`
Kann eine *IOException* werfen.

ObjectInputStream-Klasse:

`ObjectInputStream(...)`
Kann eine *IOException* werfen.

`readObject(...)`
Liest ein Objekt aus dem *ObjectInputStream* ein. Diese Objekte sind die Instanzen der gespeicherten Kommandos (*Commands*). Beim erneuten Aufruf wird das nächste Objekt eingelesen. Existiert kein weiteres Objekt, so wird NULL zurückgegeben. Die Methode kann eine *IOException* und eine *ClassNotFoundException* werfen.

`close(...)`
Schließt den Input Stream und muss aufgerufen werden um die Datei-Ressource freizugeben. Kann eine *IOException* werfen.

Hardware-Klasse:

`readFromHardware(...)`
Ist eine abstrakte Methode. Kann eine *UnsupportedOperationException* werfen.

`writeToHardware(...)`
Ist eine abstrakte Methode. Kann eine *UnsupportedOperationException* werfen.

`getType(...)`
Ist eine abstrakte Methode.

ADC-Klasse:

`getType(...)`
Liefert den Hardwaretyp ADC zurück.

`read(...)`
Gibt den gelesenen Wert zurück. Kann eine *HardwareException* werfen.

DAC-Klasse:

`getType(...)`
Liefert den Hardwaretyp DAC zurück.

`write(...)`
Setzt den übergebenen Wert am DAC. Kann eine *HardwareException* werfen.

GPIO-Klasse:

`getType(...)`
Liefert den Hardwaretyp GPIO zurück.

`write(...)`
Entnehmen Sie den Ablauf den Sequenzdiagrammen. In Fehlerfällen wird eine *HardwareException* geworfen.

`read(...)`
Entnehmen Sie den Ablauf den Sequenzdiagrammen. In Fehlerfällen wird eine *HardwareException* geworfen.

`readFromHardware(...)`
Kann eine *UnsupportedOperationException* werfen.

`writeToHardware(...)`
Kann eine *UnsupportedOperationException* werfen.

Broker-Klasse:

`pushMessage(...)`
Fügt die übergebene Nachricht der Nachrichtenliste hinzu und benachrichtigt alle registrierten Beobachter.

`register(...)`
Registriert den übergebenen Beobachter. Ist der übergebene Parameter keine Instanz, so wird keine Aktion ausgeführt.

`notifyObservers(...)`
Benachrichtigt alle Beobachter über eine neue Nachricht. Die Beobachter erhalten die eindeutige Identifikationsnummer der Nachricht.

`getMessage(...)`
Liefert jene Nachricht zurück deren Identifikationsnummer mit der angeforderten übereinstimmt. Diese Methode gibt einen Container mit der gefundenen Nachricht als Inhalt oder einen leeren Container, wenn keine Nachricht gefunden wurde, zurück.

Optional-Klasse:

Ein Optional ist ein Container, in dem ein Objekt enthalten sein kann. Ist ein Objekt enthalten ist liefert die Methode *isPresent()* TRUE und *get()* liefert das enthaltene Objekt.

`empty(...)`
Liefert eine leere Optional-Instanz zurück. In dem Container ist kein Objekt enthalten.

`get(...)`
Ist ein Objekt enthalten liefert die Methode das Objekt zurück, anderenfalls wird eine *NoSuchElementException* geworfen.

`isPresent(...)`
Liefert TRUE wenn ein Objekt enthalten ist anderenfalls FALSE.

`of(...)`
Gibt ein Optional zurück in dem das übergebene Objekt enthalten ist. Ist der übergebene Parameter NULL, so wird eine *NullPointerException* (Laufzeit-Exception!) geworfen.

Klassendiagramm:

Abbildung 3 - CommandExecutor Klassendiagramm

Beispiel 1

12.12.2018

Sequenzdiagramm 1:

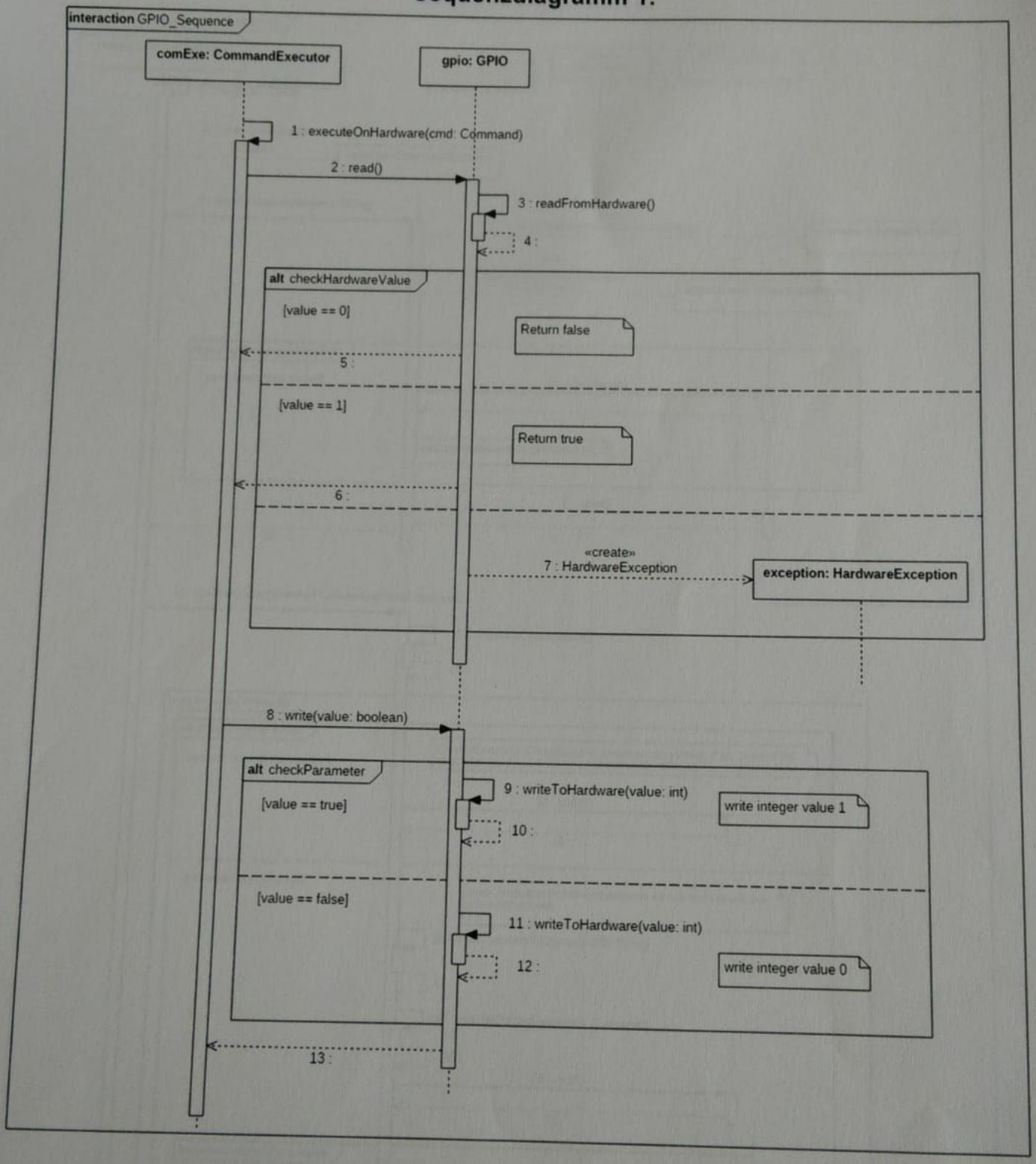

Abbildung 4 – Sequenzdiagramm 1

Sequenzdiagramm 2:

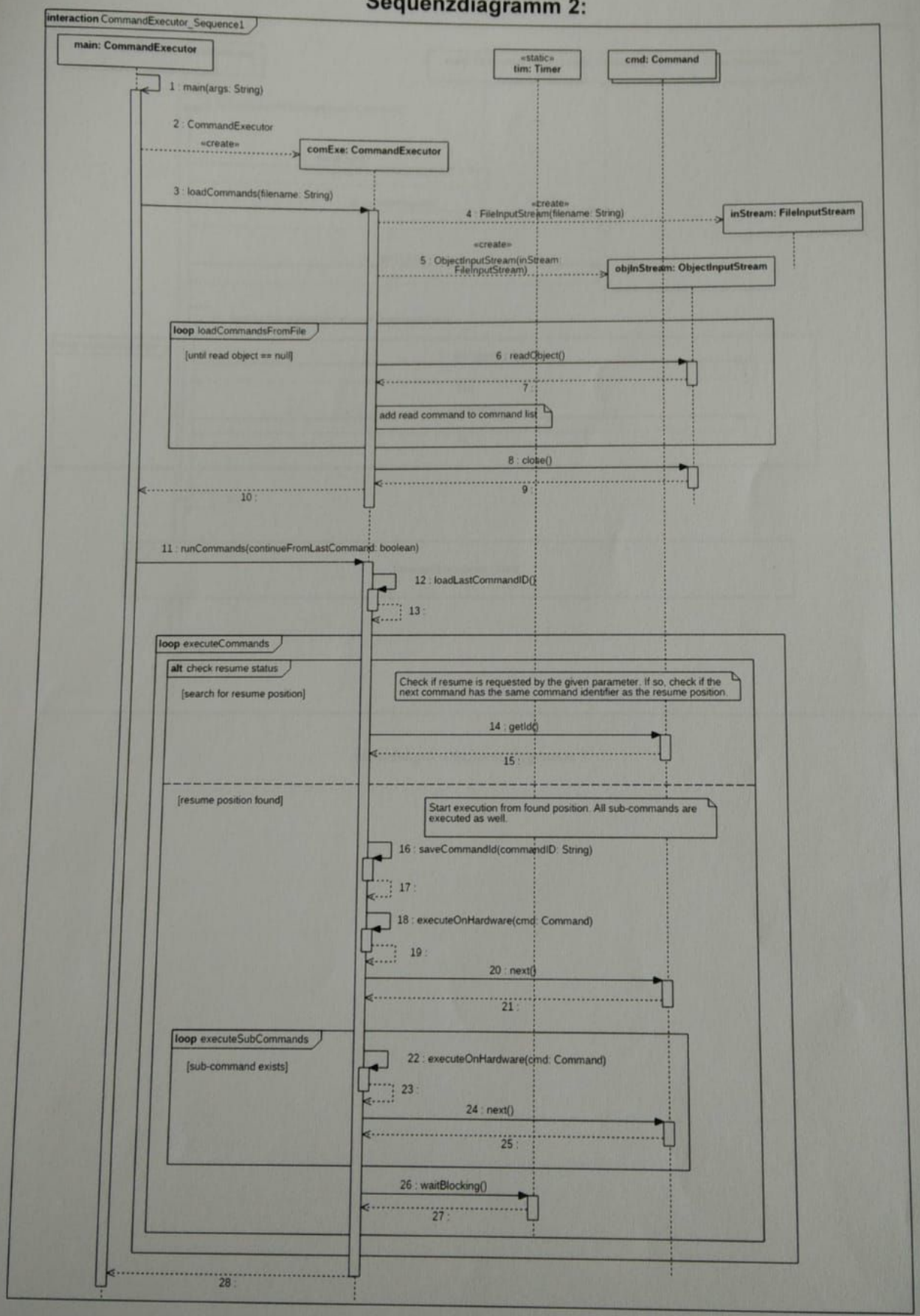

Abbildung 5 – Sequenzdiagramm 2

Sequenzdiagramm 3:

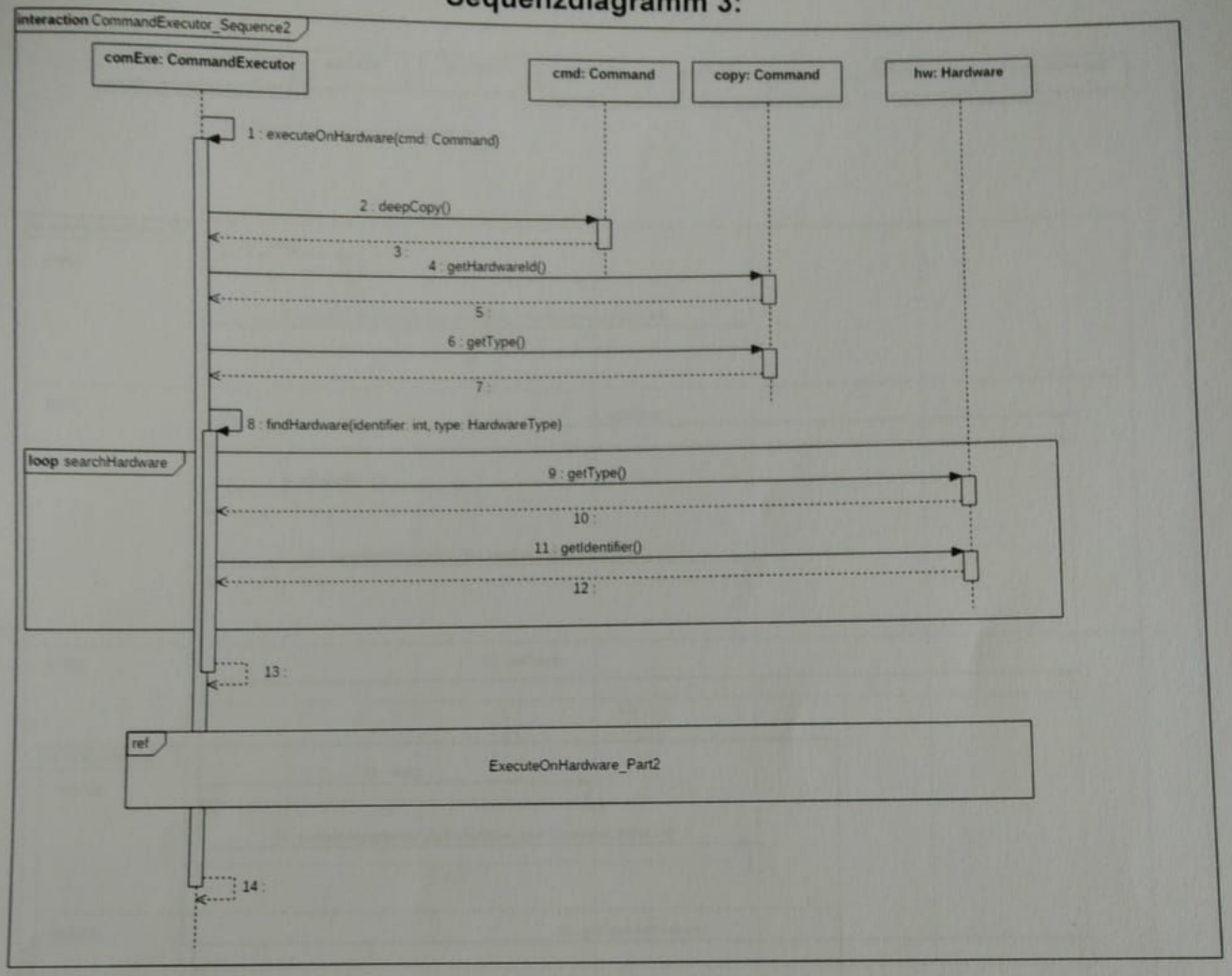

Abbildung 6 –Sequenzdiagramm 3

Sequenzdiagramm 4:

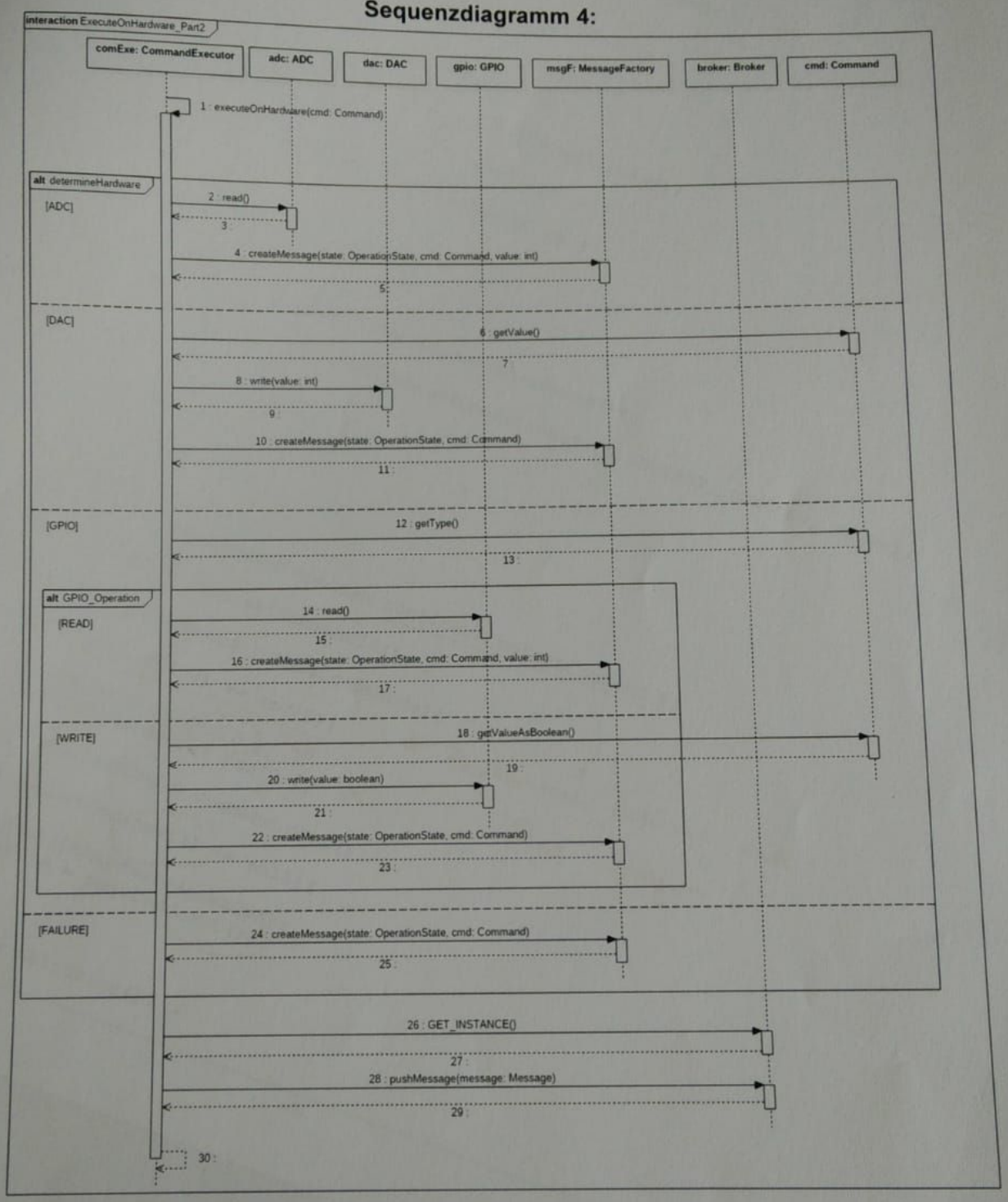

Abbildung 7 –Sequenzdiagramm 4

Name: _____ Mat.Nr. : _____

Beispiel 3 (20 Punkte) 9/20

Beantworten Sie die vier nachfolgenden Fragen (je max. 5 Punkte) **kurz** aber dennoch **prägnant**, d.h. **vollständig**:

3,5

A) Erklären Sie alle Elemente der Methodensignatur. Können Sie Methoden auch ohne Objektreferenzen aufrufen, und wenn ja wie? Wie werden in Java Parameter übergeben und welchen Einfluss hat dies beim Zugriff auf die übergebenen Parameter?

Bsp 3 ⇒ Ausdrucke !!

1

B) Was definiert ein Typ in OOP? Wie können Ausprägungen (Instanzen) von Typen miteinander interagieren? Wodurch werden die Interaktionsmöglichkeiten festgelegt? Welche Auswirkung hat eine Typhierarchie hierauf?

Name: _____

Mat.Nr. : _____

C) Zeigen und Erklären Sie den Ablauf bei der Objekterzeugung und Zuweisung auf eine Variable. Was wird in der Variable abgespeichert? Wie hängt die Erzeugung eines Objektes mit der Vererbungshierarchie zusammen?

D) Nennen Sie fünf Arten des Testens, außer Black-Box und White-Box-Testen. Erklären und beschreiben Sie die von Ihnen genannten Test-Arten.

Datum: 12.12.2018
Ort: DCEGD19