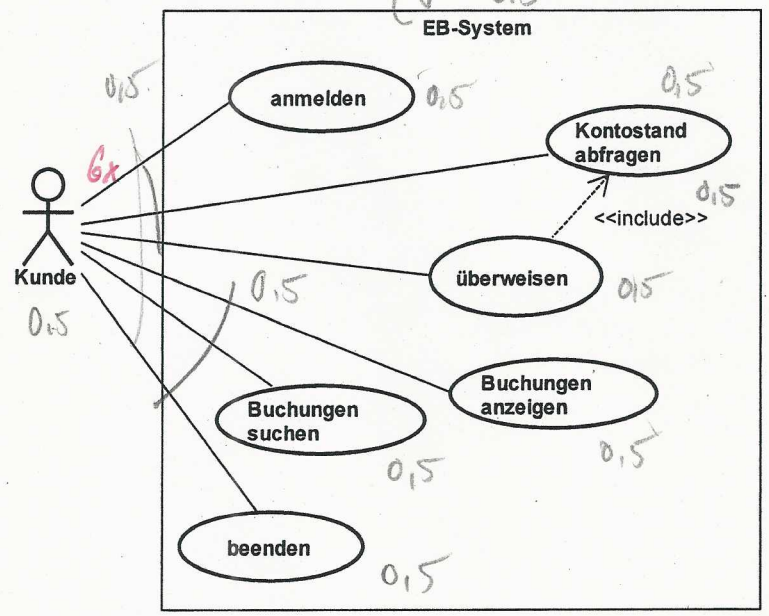
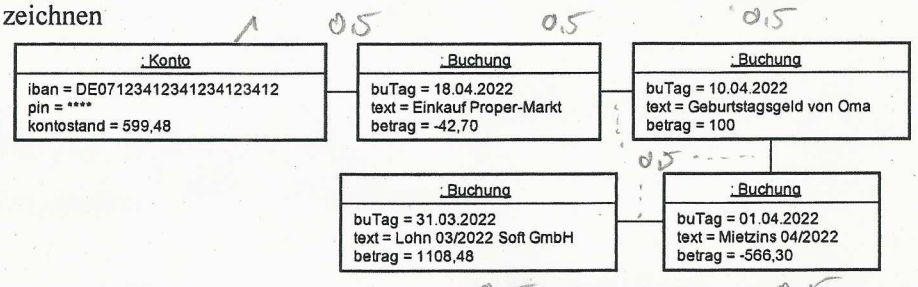
BE-Verteilung Master für Korrektur (Haug) - OOSE 10.05.2022

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.1	entwickeln, zeichnen  entwickeln zeichnen	2	3	
1.2	beschreiben Als Datenkapselung wird der kontrollierte Zugriff auf Attribute und Methoden von Klassen bezeichnet. Klassen sollen den internen Zustand anderer Klassen nicht in unkontrollierter Weise lesen oder ändern können. Eine Klasse hat eine Schnittstelle, die darüber bestimmt, wie mit der Klasse agiert werden kann. Die im Klassendiagramm enthaltenen Zugriffsrechte sind: public (+): Zugreifbar für alle Objekte. private (-): Zugreifbar nur innerhalb der eigenen Klasse. erläutern Die Attribute `pin` und `kontostand` der Klasse `Konto` sind `private` und somit von außerhalb der Klasse nicht sichtbar. Der Kontostand kann mit einer entsprechenden get-Methode abgefragt werden, eine set-Methode fehlt hingegen. Der Kontostand kann damit nur über die öffentlichen Methoden `einzahlen()` und `überweisen()` geändert werden. Das Attribut `pin` ist von außen ausschließlich über die Schnittstelle `login()` erreichbar.	2	2	
1.3	zeichnen  	4		

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.4	implementieren	2	6	6

```
public class Konto {
  private String iban;
  private int pin;
  private double kontostand;
  private Buchung neueste;

  public Konto(String iban) {
    this.iban = iban;
    pin = EBVerwaltung.generierePin();
    kontostand = 0;
    neueste = null;
  }

  public boolean login(int pin) {
    return this.pin == pin;
  }

  public void einzahlen(String text, double betrag) {
    kontostand += betrag;
    hinzufuegenBuchung(new Buchung(text, betrag));
  }

  public boolean ueberweisen(Konto empfaenger, String text,
                             double betrag){
    boolean ok = false;
    if (betrag <= kontostand) {
      empfaenger.einzahlen(text, betrag);
      kontostand -= betrag;
      hinzufuegenBuchung(new Buchung(text, (betrag*-1)));
      ok = true;
    }
    return ok;
  }

  private void hinzufuegenBuchung(Buchung b) {
    b.setVorherige(neueste);
    neueste = b;
  }

  public String zeigeBuchungen() {
    String s = "Konto IBAN\t" + iban + "\tKontostand\t" +
       kontostand + " EUR\n" +
       "Buchungstag\tBuchungstext\tBetrag in EUR\n";
    Buchung b = neueste;
    while(b != null)  {
      s += b.toString()+ "\n";
      b = b.getVorherige();
    }
    return s;
  }

  public List<Buchung> sucheBuchungen(String begriff) {
    List<Buchung> gesuchte = new List<>();
    Buchung b = neueste;
    while(b != null) {
      if(b.getText().contains(begriff)) {
```

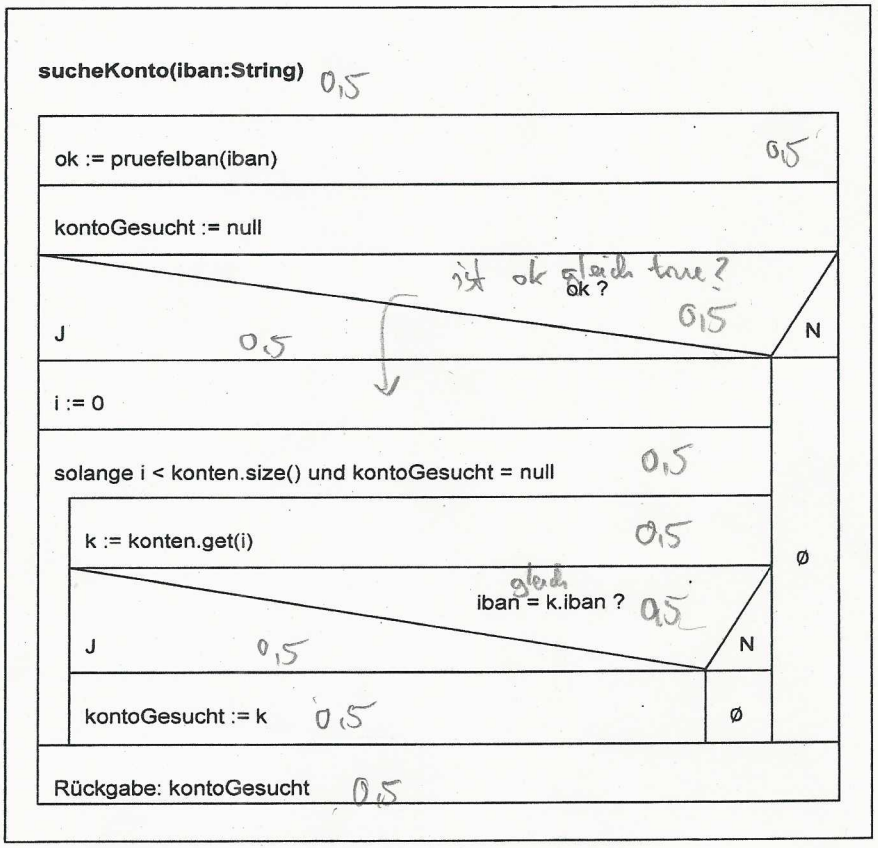
Aufg.	erwartete Leistungen	BE I	BE II	BE III

```
        gesuchte.add(b);
      }
      b = b.getVorherige();
    }
    return gesuchte;
  }
}

public class Buchung {
  private Date datum;
  private String text;
  private double betrag;
  private Buchung vorherige;

  public Buchung(String text, double betrag) {
    this.datum = new Date();
    this.text = text;
    this.betrag = betrag;
  }
  public String toString() {
    String s = datum.toString() + "\t" + text + "\t" +
               betrag;
    return s;
  }
}
```

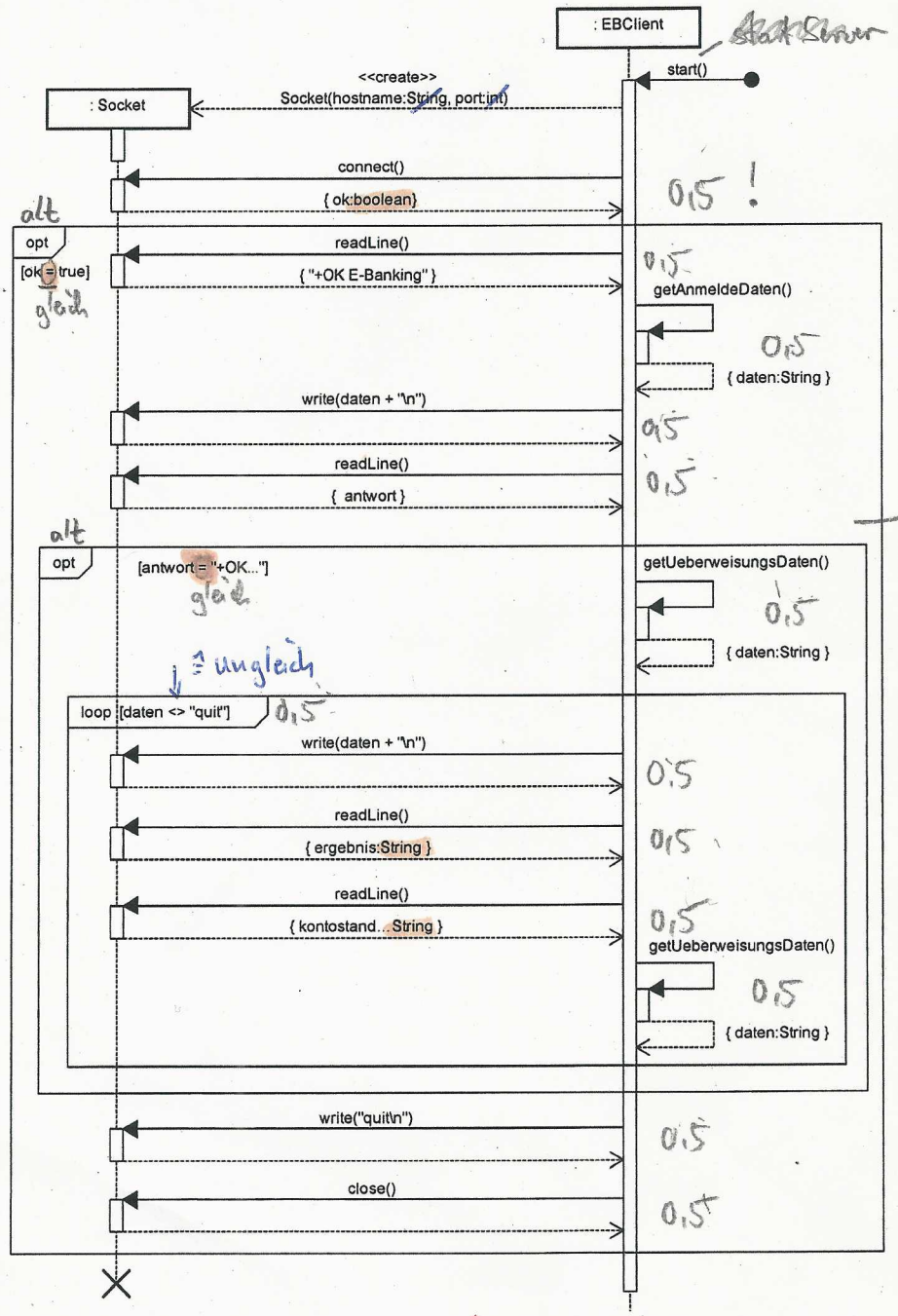
1.5 entwickeln

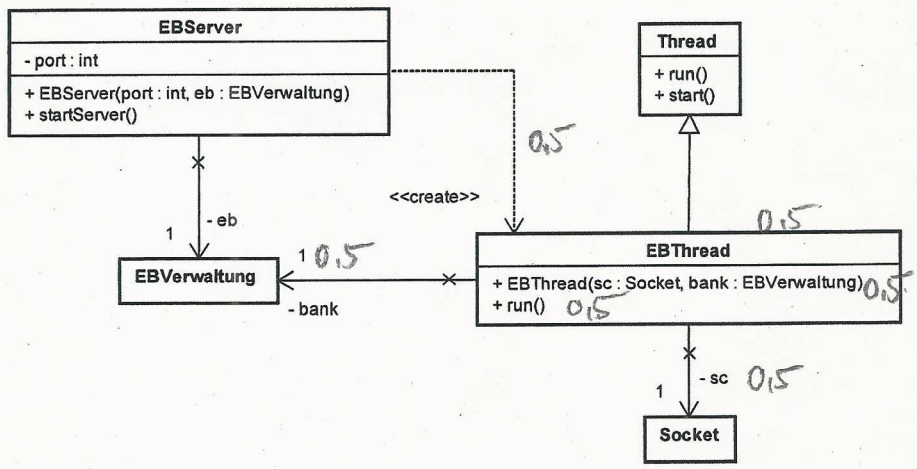


Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.6.1	erläutern Das Client-Server-Modell beschreibt die Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerks zu verteilen. Die Aufgaben werden von Programmen erledigt, die in Clients und Server unterteilt werden. Der EB-Client fordert die Erledigung einer Aufgabe vom EB-Server an, wie das Anmelden und Durchführen von Überweisungen. Der Client ist aktiv. Der Server ist passiv und wartet auf Verbindungsanfragen. Er bearbeitet die Anfrage und gibt eine Rückmeldung über den Erfolg der Ausführung. Ein Dienst ist eine solche festgelegte Aufgabe, die der Server anbietet und der Client nutzt. Die Regeln der Kommunikation für einen Dienst (Kommandos, die Bedeutung der zwischen Server und Client ausgetauschten Daten und Formate) werden durch ein Protokoll festgelegt. Das Protokoll ist spezifisch für den jeweiligen Dienst.		4	

1.6.2	implementieren
	`public class EBServer {`
	`  private ServerSocket server = null;`
	`  private EBVerwaltung eb;`
	`  public EBServer(int port, EBVerwaltung eb){`
	`  this.port = port;`
	`  this.eb = eb;`
	`  }`
	`  public void startServer(){`
	`  server = new ServerSocket(port);`
	`  while(true){`
	`  Socket sc = server.accept();`
	`  sc.write("+OK E-Banking\n");`
	`  String anforderung = sc.readLine();`
	`  String[] werte = anforderung.split(";");`
	`  Konto k = null;`
	`  k = eb.anmelden(werte[1], Integer.parseInt(werte[2]));`
	`  if(k == null) {`
	`  sc.write("-ERR Login fehlgeschlagen!\n");`
	`  } else {`
	`  sc.write("+OK Willkommen\n");`
	`  anforderung = sc.readLine();`
	`  while (!anforderung.equals("quit")){`
	`  werte = anforderung.split(";");`
	`  Konto anderesKonto = eb.sucheKonto(werte[1]);`
	`  if(anderesKonto != null ){`
	`  if(eb.ueberweisen(k, anderesKonto, werte[2],`
	`  Double.parseDouble(werte[3])))`
	`  sc.write("+OK Überweisung erfolgt\n");`
	`  else`
	`  sc.write("-ERR Überweisung nicht erfolgt\n");`
	`  } else {`
	`  sc.write("-ERR Ungültige IBAN\n");`
	`  }`
	`  sc.write("Aktueller Kontostand: " +`
	`  k.getKontostand() + " EURO\n");`
	`  anforderung = sc.readLine();`
	`  }`
	`  }`
		4	2	3

Aufg.	erwartete Leistungen	BE I	BE II	BE III
0,5	`sc.close();` 0,5 `}` `}` `}`			
1.6.3	entwickeln, zeichnen			
	entwickeln zeichnen	2	3	3



Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.7.1	beschreiben Die sequentielle Ausführung von Anweisungen eines Programms stellt einen Prozess dar, für den ein eigener Speicherbereich reserviert ist und der vom Betriebssystem verwaltet, gestartet und angehalten wird. Ein Thread ist ein einzelner in sich geschlossener Steuerfluss innerhalb eines Prozesses. Jeder Prozess besitzt einen Haupt-Thread, mit dem das Programm gestartet wird (main). Mehrere neue Threads können vom Programm gestartet werden. Diese Threads laufen dann alle quasi-parallel ab, besitzen jeweils einen eigenen Zustand mit Befehlszähler und Stack, arbeiten aber im Gegensatz zu Prozessen auf demselben Speicherbereich im Arbeitsspeicher.	2		
	erläutern Mehrere Threads können über die Kontomethoden auf die Variable `kontostand` eines Konto-Objekts gleichzeitig zugreifen. Es darf aber nicht vorkommen, dass ein Thread bereits aus einem Objekt liest, während ein anderer Thread noch Daten desselben Objekts ändert. Somit könnte eine Überweisung erfolgen obwohl keine ausreichende Deckung vorliegt.		1	
	benennen Das wird verhindert, indem die Methoden `ueberweisen()` und `einzahlen()` synchronisiert werden.		1	
1.7.2	entwickeln, zeichnen  			
	entwickeln zeichnen	1	1	1
	Summe 60	**19**	**26**	**15**