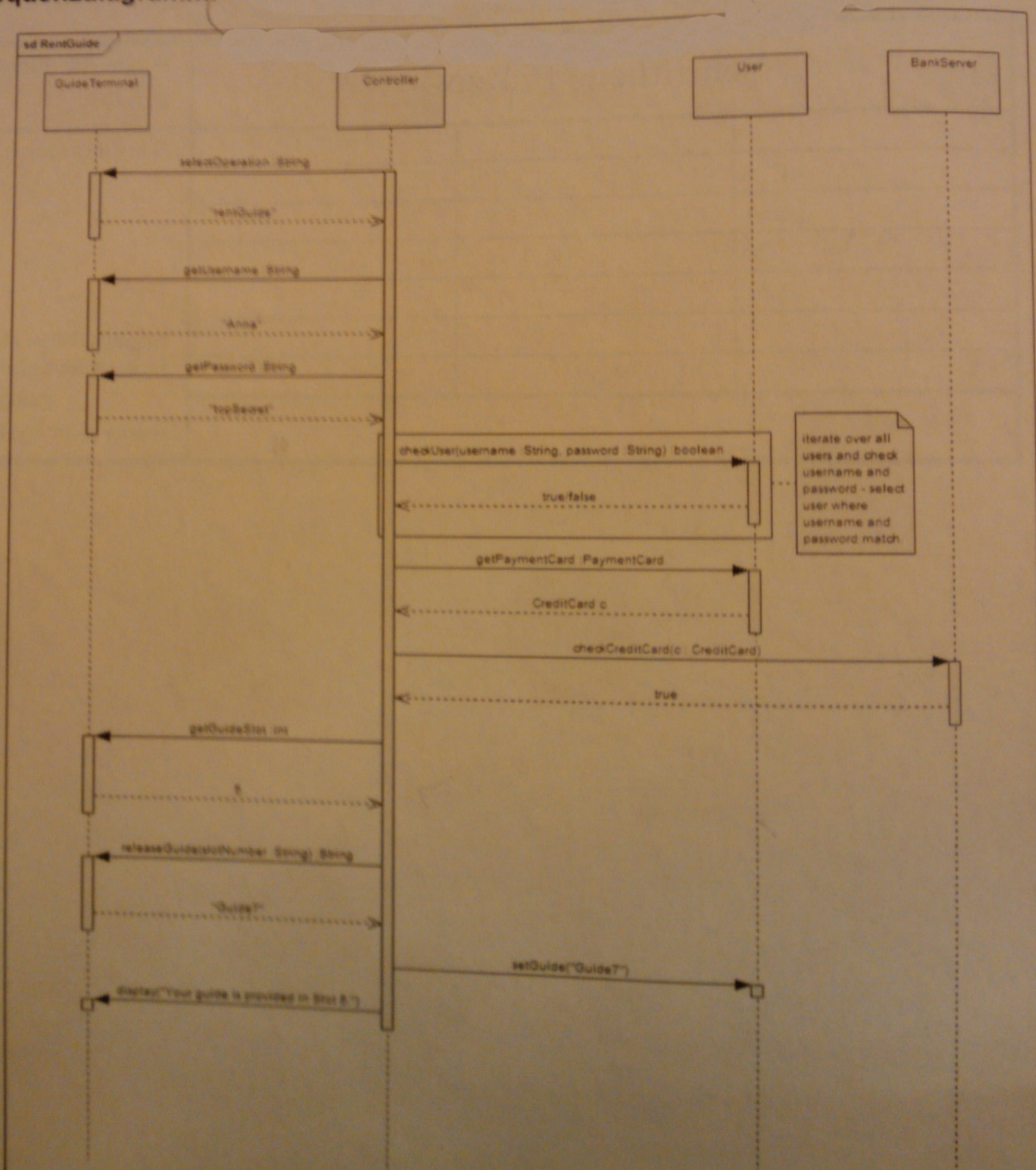
Beispiel 1 (55 Punkte)

Gegeben sind ein **Klassen**- und ein **Sequenzdiagramm** eines Software-Entwurfs für eine Museums-Guide Verleih-Applikation. **Implementieren Sie die Klassen (Methoden und Attribute) entsprechend dem Sequenz-, dem Klassendiagramm und dem angegebenen Code der Klasse `"Controller"`.**

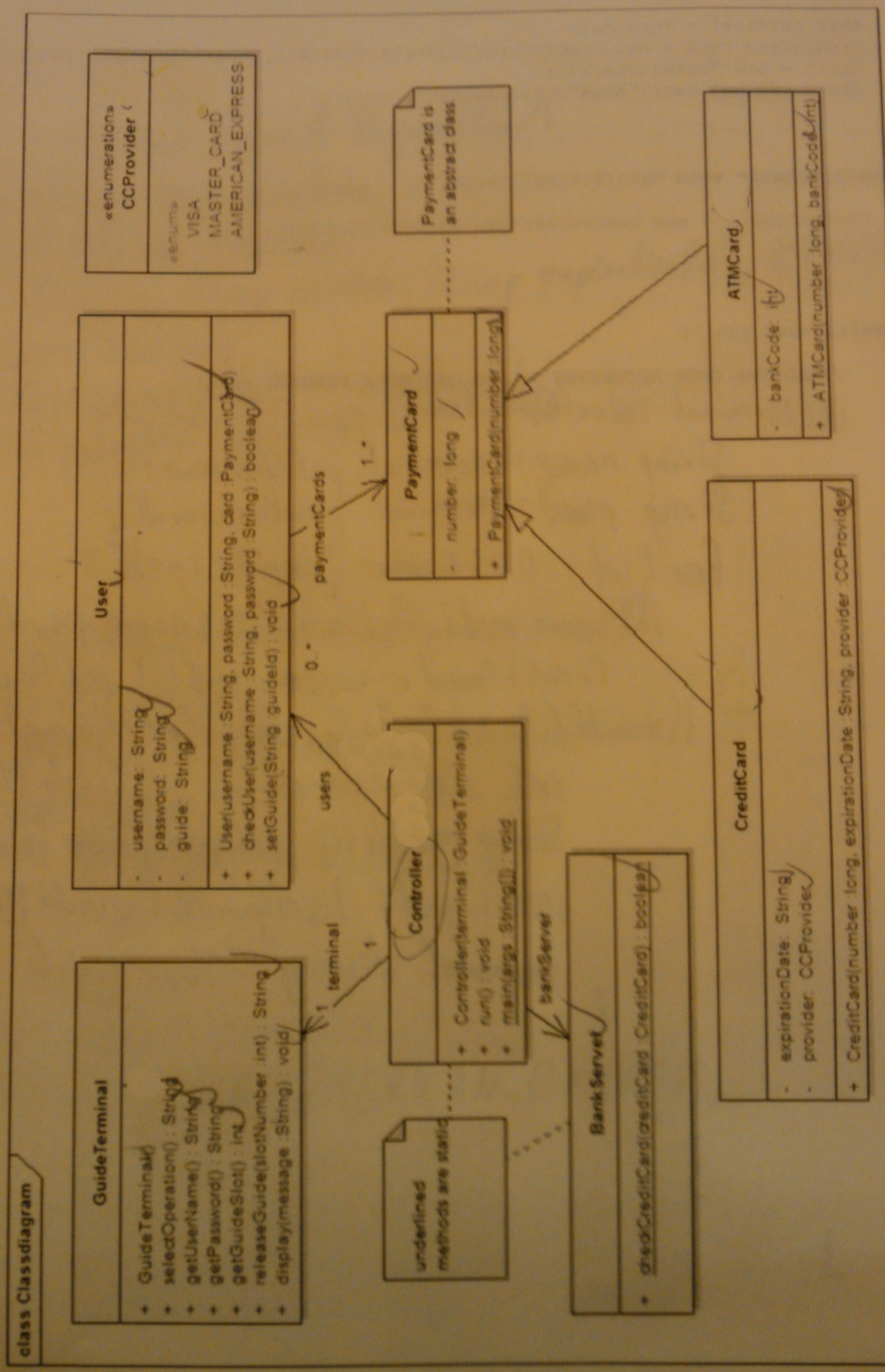
Das Sequenzdiagramm beschreibt den Vorgang einer Guide-Entlehnung bei einem Terminal (GuideTerminal). Das Objekt der Klasse `GuideTerminal` kapselt die Schnittstelle zum Anwender. Eingaben eines Benutzers werden darin „simuliert", d.h. es reicht, wenn z.B. die Methode `GuideTerminal.getOperation()` den String „rentGuide" liefert – Sie brauchen **keine** interaktive Benutzereingabe zu implementieren. Überprüfen Sie trotzdem in der `Controller` Klasse, ob die erwarteten Rückgabewerte zurückkommen.

Sequenzdiagramm:



Klassendiagramm

Assoziationen mit einem Pfeil (gerichtete Assoziationen) im Klassendiagramm bedeuten, dass jene Klasse beim Pfeilursprung eine Referenz (gespeichert in einer Instanzvariable) zur Klasse am Pfeilende besitzt. Diese Referenzen werden entsprechend der Kardinalitäten auf der Relation entweder als Objekt oder als Liste von Objekten gespeichert. Bitte beachten Sie, dass `PaymentCard` eine abstrakte Klasse ist und unterstrichene Methoden statisch (d.h., `static`) sind. Variablen und Methoden sind entweder `private`, gekennzeichnet durch ein „-“ im Diagramm, oder `public`, gekennzeichnet durch ein „+“.



Beispiel 2 (25 Punkte)

a) (4 Punkte)

Gegeben sind die drei Klassen `DryFruit`, `Apple` und `Pear` sowie die Schnittstelle `IFruit`.

```
public abstract class DryFruit { ... }

public class Apple extends DryFruit {
   ...
}

public interface IFruit { ... }

public class Pear implements IFruit {
   ...
}
```

```
public class Test {

  public static void main( String[] args ) {
    ...
    IFruit c = new Pear();      // C_1
    DryFruit d = new Apple();   // C_2
    Apple a  = new DryFruit();  // C_3
    IFruit b = new IFruit();    // C_4
    ...
  }
}
```

Ist das Kreieren der vier Instanzen in der Klasse `Test` erlaubt?
Kreuzen Sie **an** und **begründen** Sie Ihre **Antworten** (Antworten **ohne Begründung** werden **nicht gewertet**!).

Codezeile:	Erlaubt? Ja	Nein	Begründung:
C_1:			
C_2:			
C_3:			
C_4:			

b) (4 Punkte)

Analysieren Sie die folgenden Klassen `Calc` und `SciCalc`. Die `main`-Methode der Klasse `Test` erstellt Objekte und sendet den Objekten Nachrichten, die gewisse Ausgaben bewirken. Tragen Sie die jeweiligen Ausgaben zu den gefragten Zeitpunkten (T1 bis T4, siehe Kommentare in `main`) in der Tabelle ein, genauso wie es für den Zeitpunkt T0 vorgegeben ist.

```
public class Calc {

  public void show( String val ) {
    System.out.println( "BASIC" + val );
  }

  public void show( int val ) {
    if ( val == 0 )
        System.out.println( "null" );
    else
        System.out.println( "val: "+val );
  }
}
```

```
public class SciCalc extends Calc {

  public void show( String val ) {
    System.out.println( "SCI" + val );
  }

  public void show( boolean val ) {
    if ( ! val )
        System.out.println( "falsch" );
    else
        System.out.println( "wahr" );
  }
}
```

```
public class Test {
    public static void main( String[] args ) {

        Calc a = new Calc();
        SciCalc b = new SciCalc();

        System.out.println( "START" );     // T0
        b.show( false );                    // T1
        a.show( 5 );                        // T2
        a.show( "TRUE" );                   // T3
        a = b;
        a.show( "AMAZING" );                // T4
    }
}
```

Ausgaben:

T0: START T1: T2: T3: T4:

Gegeben sind folgende Schnittstellen und Klassen:

```
public interface Int_1 {
    ...
}

public interface Int_2 {
    ...
}

public abstract class AbstractC {
    ...
}

public class Class_1 {
    ...
}

public class Class_2 {
    ...
}
```

Kreuzen Sie **an**, ob die jeweilige Deklaration in Java **erlaubt** ist?

	Ja	Nein
`public class PL extends Int_2 implements Class_1 { ... }`		
`public class EA extends AbstractC { ... }`		
`public class SE extends Class_1 { ... }`		
`private class DONT implements AbstractC { ... }`		
`public class CO extends Class_2 implements Int_1 { ... }`		
`public class PT implements Int_1, Int_2 { ... }`		
`private class FR extends AbstractC implements Int_1 { ... }`		
`public class OM implements Class_1 { ... }`		
`public class TO extends Int_1 { ... }`		
`public class UR extends Int_1 implements AbstractC { ... }`		
`public class NEIGH implements Int_1 { ... }`		
`private class BOOR extends Class_1, Class_2 { ... }`		

d) (5 Punkte)

```
public class Person {...}

public class Passenger extends Person {...}

public class Employee extends Person {...}

public class Pilot extends Employee {...}

public class Steward extends Employee {...}
```

```
public class Controller {

    public void doSomething( Employee e ) {
        ...
    }
}
```

```
public class Test {

    public static void main( String[] args ) {
        Person a = new Person();
        Passenger b = new Passenger();
        Employee c = new Employee();
        Pilot d = new Pilot();
        Steward e = new Steward();
        Controller u = new Controller();

        u.doSomething( e );      // C_1
        u.doSomething( d );      // C_2
        u.doSomething( c );      // C_3
        u.doSomething( b );      // C_4
        u.doSomething( a );      // C_5
    }
}
```

Welcher der Aufrufe der Methode `User.doSomething()` in der Klasse `Test` ist erlaubt?
Kreuzen Sie **an** und **begründen Sie Ihre Antworten** (Antworten **ohne Begründung** werden **nicht gewertet**!).

Codezeile:	Erlaubt? Ja	Erlaubt? Nein	Begründung:
C_1:			
C_2:			
C_3:			
C_4:			
C_5:			

1) Ein Interface ist eine Sammlung von Methodendefinitionen ohne Implementierung.

2) Instanzvariablen definieren das Verhalten und Instanzmethoden den Zustand eines Objekts.

3) Polymorphismus ist ein Designpattern.

4) Patterns sind strukturierte Beschreibungen für Lösungsansätze wiederkehrender Problemstellungen.

5) Das Singleton Pattern gewährleistet, dass alle Methoden der Singleton-Klasse global verfügbar sind.

6) Eine abstrakte Klasse kann nur genau eine Instanz zur Laufzeit haben.

7) Abstrakte Klassen sind nicht Teil der Klassenhierarchie.

8) Eine Klasse kann mehrere Konstruktoren haben.

9) Eine abstrakte Klasse kann auch mehr als ein Interface implementieren.

10) Mit Hilfe einer objektorientierten Programmiersprache wie Java kann man eigene Datentypen definieren.