Beispiel 1 (60 Punkte)

Beispiel 1a) (30 Punkte)

Implementieren Sie unter Verwendung des Observer-Patterns eine Klasse zur *Verwaltung* **(10 Punkte)** einer nichtnegativen ganzzahligen Temperatur (in Grad Celsius) sowie drei Klassen zur *Anzeige*, welche diese Temperatur beobachten und auf der Konsole als String die Ziffernfolge auf der Celsius- **(2 Punkte)**, der Fahrenheit- **(3 Punkte)** sowie der Kelvinskala **(3 Punkte)** ausgeben. Erstellen Sie auch alle benötigten *Interfaces* **(2 Punkte)** die im Observer-Pattern definiert sind.

Jede Änderung der Temperatur soll durch Ausgaben auf der Konsole dokumentiert werden.

Ergänzen Sie diese Implementierung um eine *Steuerungsklasse* **(5 Punkte)**, welche folgenden Ablauf steuert:

1. Erzeugen Sie ein Objekt zur Verwaltung der Temperatur.
2. Setzen Sie die Temperatur auf 1 °C.
3. Erzeugen Sie ein Objekt zur Anzeige von Celsiuswerten.
4. Erzeugen Sie ein Objekt zur Anzeige von Fahrenheitwerten.
5. Registrieren Sie beide Objekte als Observer.
6. Setzen Sie die Temperatur auf 5 °C.
7. Erzeugen Sie ein Objekt zur Anzeige von Kelvinwerten.
8. Registrieren Sie dieses Objekt als Observer.
9. Setzen Sie die Temperatur auf 20 °C.

Die Steuerungsklasse soll auch die `main`-Methode beinhalten.
Listen Sie auch den zu erwartenden Output der Observer (nachdem diese benachrichtigt wurden) in richtiger Reihenfolge auf **(5 Punkte)**.

Hinweise:

Senden Sie beim Benachrichtigen der Observer die geänderte Temperatur als Parameter mit.

Um eine Temperatur x in °C in eine Temperatur y in °F umzurechnen, verwenden Sie folgende Formel: $y = (x \cdot 9 / 5) - 32$. Um eine Temperatur x in °C in eine Temperatur z in K umzurechnen, verwenden

Um eine Temperatur x in °C in eine Temperatur y in °F umzurechnen, verwenden Sie folgende Formel: $y = (x * 9 / 5) - 32$. Um eine Temperatur x in °C in eine Temperatur z in K umzurechnen, verwenden Sie folgende Formel: $z = x + 273,15$. Beachten Sie, dass zur Speicherung der umgerechneten Ergebnisse der Typ `float` notwendig ist.

Beispiel 1b) (10 Punkte)

Implementieren Sie eine Klasse *Rechteck*:

- Rechtecke besitzen zwei ganzzahlige Attribute, welche die Seiten (a, b) darstellen (**1 Punkt**).
- Erstellen Sie einen Standardkonstruktor, welcher beide Attribute auf 0 setzt (**1 Punkt**).
- Erstellen Sie einen Konstruktor, der Werte für die beiden Attribute entgegen nimmt und diese entsprechend setzt (**1 Punkt**).
- Schreiben Sie eine Methode `groesserAls`, die zwei Rechtecke miteinander vergleicht (**2 Punkte**). Die Methode retourniert *true*, wenn das als Parameter übergebene Rechteck größer ist. Im umgekehrten Fall retourniert die Methode *false*. Ein Rechteck ist genau dann größer, wenn eine Seite größer und die andere Seite nicht kleiner ist (im Vergleich zu den beiden Seiten des anderen Rechtecks).
- Schreiben Sie eine Methode `ausgabe`, welche die Längen der beiden Seiten auf der Konsole ausgibt (**1 Punkt**).
- Schreiben Sie eine Methode `tausche`, welche die beiden Seiten des Rechtecks vertauscht (**3 Punkte**).
- Schreiben Sie eine Methode `flaeche`, welche den Flächeninhalt ($A = a \cdot b$) des Rechtecks berechnet und als `int`-Wert zurückgibt (**1 Punkt**).

Beispiel 1c) (20 Punkte)

Gegeben sind ein **Sequenz-** und ein **Klassendiagramm** eines Software-Entwurfs. Basierend auf diesen Diagra folgende Aufgabe zu lösen:

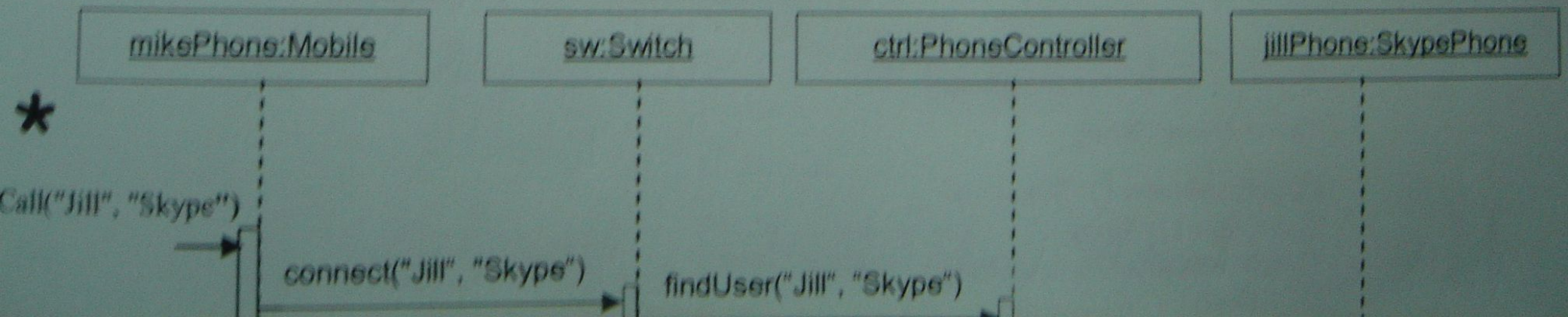
Implementieren Sie die Klassen (Methoden und Attribute) entsprechend dem Sequenz-, dem Klassendiagramm und dem angegebenen Code der Klasse "Angabe" (in dieser Klasse wird das Progra initialisiert).

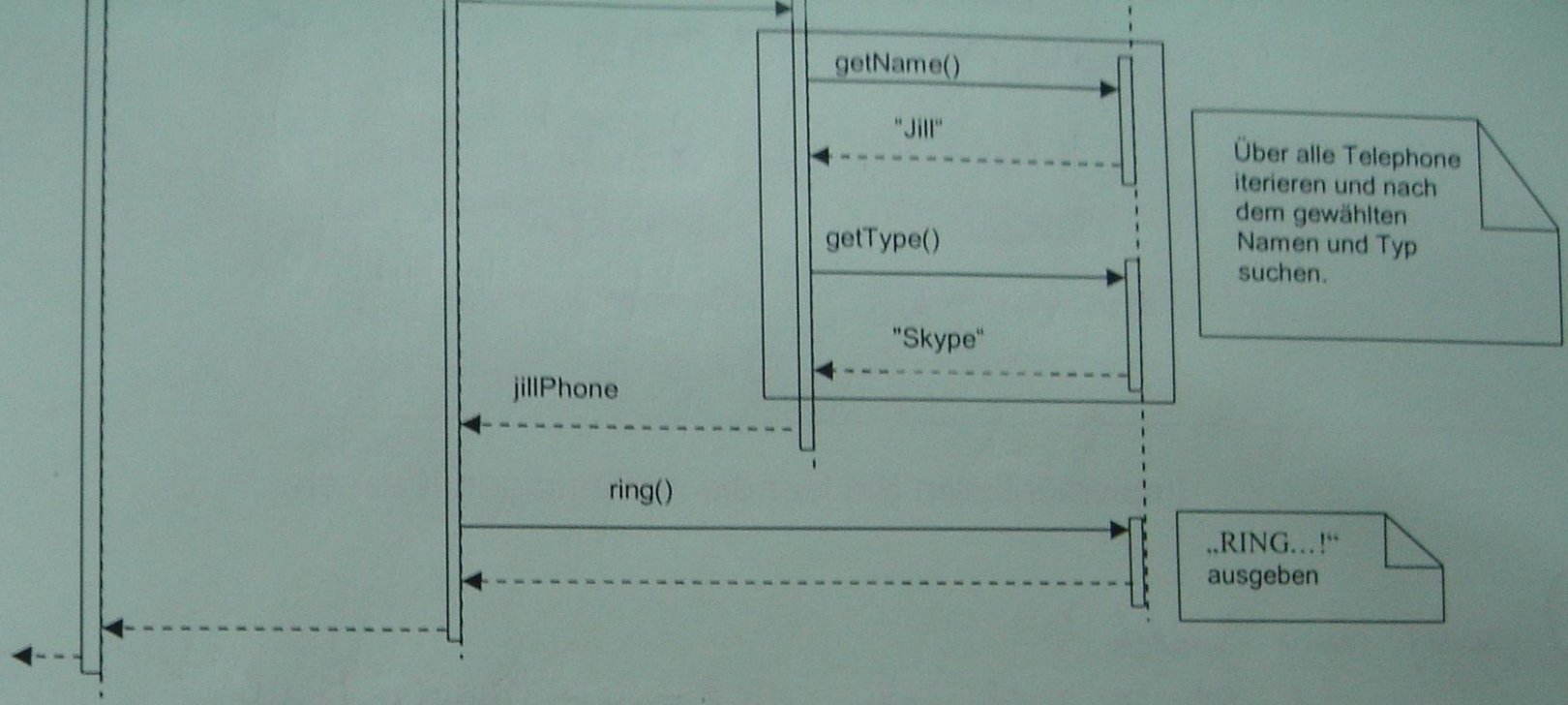
Der Ablauf im Sequenzdiagramm (markiert mit "*") beginnt in der Methode `startCall()` der Klasse

Das Sequenzdiagramm beschreibt den Vorgang eines Rufaufbaus in einem Telefonnetz. Der Mobiln „Mike" ruft „Jill" am Skypetelefon an.

Hinweis: Die Klasse `Telephone` wird selbst nie instanziiert und ist deshalb als `abstract` zu implementieren. `Telephone.getType()` ist eine abstrakte Methode.

Sequenzdiagramm:






```
public class Angabe {
        public static void main(String[] args) {
                //init
                Telephone mikePhone = new Mobile();
                mikePhone.setUserName( "Mike" );
                Telephone jillPhone = new Skype Phone();
                jillPhone.setUserName( "Jill" );
                Switch sw = new Switch();
                mikePhone.setSwitch( sw );
                jillPhone.setSwitch( sw );
                Vector<Telephone> users = new Vector<Telephone>();
                users.add( mikePhone );
                users.add( jillPhone );
                PhoneController ctrl = new PhoneController();
                ctrl.setUsers( users );
                sw.setPhoneCtrl( ctrl );

                //main:makeCall
                mikePhone.startCall( "Jill", "Skype" );


        }
}
```

Beispiel 2 (10 Punkte)

a) (5 Punkte)

Analysieren Sie die folgenden Klassen **Drucker** und **DruckerNeu**. Die **main**-Methode der Klasse **Test** erstellt Objekte und sendet den Objekten Nachrichten, die gewisse Ausgaben bewirken. Geben Sie die jeweiligen Ausgaben den gefragten Zeitpunkten (T1 bis T4, siehe Kommentare in **main**) aus, genauso wie es für den Zeitpunkt T0 gezeig ist.

```
public class Drucker {

  public void zeige( String wert ) {
    System.out.println( "ALT" + wert );
  }

  public void zeige( boolean wert ) {
    if ( wert )
        System.out.println( "true" );
    else
        System.out.println( "false" );
  }
}
```

```
public class DruckerNeu extends Drucker {

  public void zeige( String wert ) {
    System.out.println( "BESSER" + wert );
  }

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

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

        Drucker d = new Drucker();
        DruckerNeu dNeu = new DruckerNeu();

        System.out.println( "BEGINN" );    // T0
                                           // T1
        d.zeige( false );                  // T2
        d.zeige( "FALSE" );
        d = dNeu;                          // T3
        d.zeige( "TRUE" );                 // T4
```

```
    d.zeige( false );                    // T0
    d.zeige( "FALSE" );                  // T1
    d = dNeu;                            // T2
    dNeu.zeige( "TRUE" );                // T3
    d.zeige( true );                     // T4
  }
}
```

Ausgaben:

T0: BEGINN T1: T2: T3: T4:

c) (5 Punkte)

Gegeben sind folgende Klassen und Schnittstellen:

```
public abstract class AbstrCl {
    ...
}

public class Class_a {
    ...
}

public class Class_b {
    ...
}

public interface Intf_c {
    ...
}

public interface Intf_d {
    ...
}
```

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

	Ja	Nein
`public class B extends Intf_d implements Class_a { ... }`	☐	☐
`public class O implements Intf_c { ... }`	☐	☐
`public class Z extends Intf_c implements AbstrCl { ... }`	☐	☐
`public class U implements AbstrCl { ... }`	☐	☐
`public class T extends Class_b implements Intf_c { ... }`	☐	☐
`public class R implements Intf_c, Intf_d { ... }`	☐	☐
`public class K extends AbstrCl implements Intf_c { ... }`	☐	☐
`public class V implements Class_a { ... }`	☐	☐
`public class X extends Class_a { ... }`	☐	☐
`public class Y extends AbstrCl { ... }`	☐	☐

Beispiel 3 (30 Punkte)

Kreuzen Sie bei den folgenden zehn Fragen WAHR an, wenn die Aussage richtig ist, und FALSCH, wenn die Aussage nicht richtig ist.

Bewertungsschema:

Für jede korrekt angekreuzte Aussage werden +3 Punkte gezählt. Wenn eine Frage nicht korrekt angekreuzt ist, werden 3 Punkte abgezogen (also -3). Wenn bei einer Frage weder WAHR noch FALSCH angekreuzt sind, gibt es 0 Punkte für die jeweilige Frage. Sie können in Summe bei Beispiel 3 nicht weniger als 0 Punkte haben, selbst wenn sich rein rechnerisch eine negative Punkteanzahl ergeben würde.

	WAHR	FALSCH
1) Das Verhalten einer Subklasse kann auch spezialisiert werden in Bezug auf die Superklasse.	O	O
2) Ein Regressionstest ist ein Whitebox-Test.	O	O
3) OOP soll die Wiederverwendung von Software erleichtern.	O	O
4) Patterns sind strukturierte Beschreibungen für Lösungsansaetze wiederkehrender Problemstellungen.	O	O
5) Beim Singleton-Pattern ist der Konstruktor private.	O	O
6) Abstrakte Klassen sind nicht Teil der Klassenhierarchie.	O	O
7) Klassen unterscheiden sich von Objekten in der unterschiedlichen Zugriffsart.	O	O
8) Ein Konstruktor einer Klasse kann mehrere Input-Parameter haben.	O	O
9) Jeder Typ ist ein Objekt.	O	O
10) Für die Ersetzbarkeit von Subtypen müssen immer die Preconditions, Postconditions aber nicht die Invarianten geprüft werden.	O	O