

Prüfung: <b>Modul 226 – Prüfung 01</b> [Version A] (ohne Unterlagen)	Zeit: <b>60 Min</b>	Klasse: <b>INF..</b>	Name & Vorname:
--	------------------------	-------------------------	-----------------

**Allg. Hinweis:** Mögliche Punktzahl = 61P

**Note = Punkte/53\*5+1**

# 1. Allgemeine Fragen (18 P.)

Im Anhang (Kapitel 4.1) finden Sie das zur Aufgabe gehörende UML-Klassendiagramm. Die folgenden Multiple-Choice-Fragen beziehen sich alle auf das besagte Klassendiagramm. Damit Sie nicht ständig blättern müssen, können Sie das Klassendiagramm im Anhang entfernen. Beschriften Sie die entfernte Seite unbedingt mit Ihrem Namen und geben sie das Blatt am Schluss mit der Prüfung ab!

**Fragen zu den Beziehungen:** **Korrekturhinweis:** **Pro Fehler -1 P** (10 P.)

Welche Aussagen sind korrekt?

- ☐ Die Klassen A und D stehen in einer Kompositionsbeziehung
- ☒ Die Klassen A und D stehen in einer gerichteten Aggregationsbeziehung
- ☐ Die Klassen D und A stehen in einer ungerichteten/bidirektionalen Assoziationsbeziehung
- ☐ Die Klassen R und D stehen in einer ungerichteten/bidirektionalen Kompositionsbeziehung
- ☒ Die Klassen D und R stehen in einer ungerichteten/bidirektionalen Assoziationsbeziehung
- ☐ Die Klassen R und N stehen in einer ungerichteten/bidirektionalen Aggregationsbeziehung
- ☐ Ein Objekt der Klasse R hat eine private Membervariable, die immer ein Objekt der Klasse W referenziert
- ☐ Die Klassen W und R stehen in einer losen Beziehung zueinander
- ☒ Die Klassen N und R stehen in einer gerichteten Assoziationsbeziehung
- ☐ Die Klassen D und R stehen in einer gerichteten Aggregationsbeziehung
- ☐ Die Klassen D und K stehen in einer losen Beziehung zueinander
- ☒ Die Klassen R und N stehen in einer gerichteten Kompositionsbeziehung
- ☐ Ein Objekt der Klasse K besitzt als Membervariable eine Referenz, die ein Objekt der Klasse D referenzieren muss.
- ☒ Ein Objekt der Klasse W hat eine private Membervariable, die immer ein Objekt der Klasse R referenziert
- ☐ Ein Objekt der Klasse D besitzt als Membervariable eine Referenz, die ein Objekt der Klasse R referenzieren kann.
- ☒ Die Klassen A und G stehen in einer losen Beziehung zueinander
- ☐ Ein Objekt der Klasse K besitzt als Membervariable eine Referenz auf ein Array, welches Referenzen von Objekten der Klasse D aufnehmen kann
- ☒ Ein Objekt der Klasse D besitzt als Membervariable eine Referenz auf ein Array, das Referenzen von Objekten der Klasse R aufnehmen kann
- ☐ Ein Objekt der Klasse N besitzt als Membervariable eine Referenz, die ein Objekt der Klasse R referenzieren kann.
- ☒ Ein Objekt der Klasse W besitzt als Membervariable eine Referenz, die ein Objekt der Klasse R referenzieren muss.
- ☒ Ein Objekt der Klasse K besitzt als Membervariable eine Referenz, die ein Objekt der Klasse D referenzieren muss.

**Fragen zur Aufrufbarkeit:**

**Korrekturhinweis: Pro Fehler -1 P**

**(8 P.)**

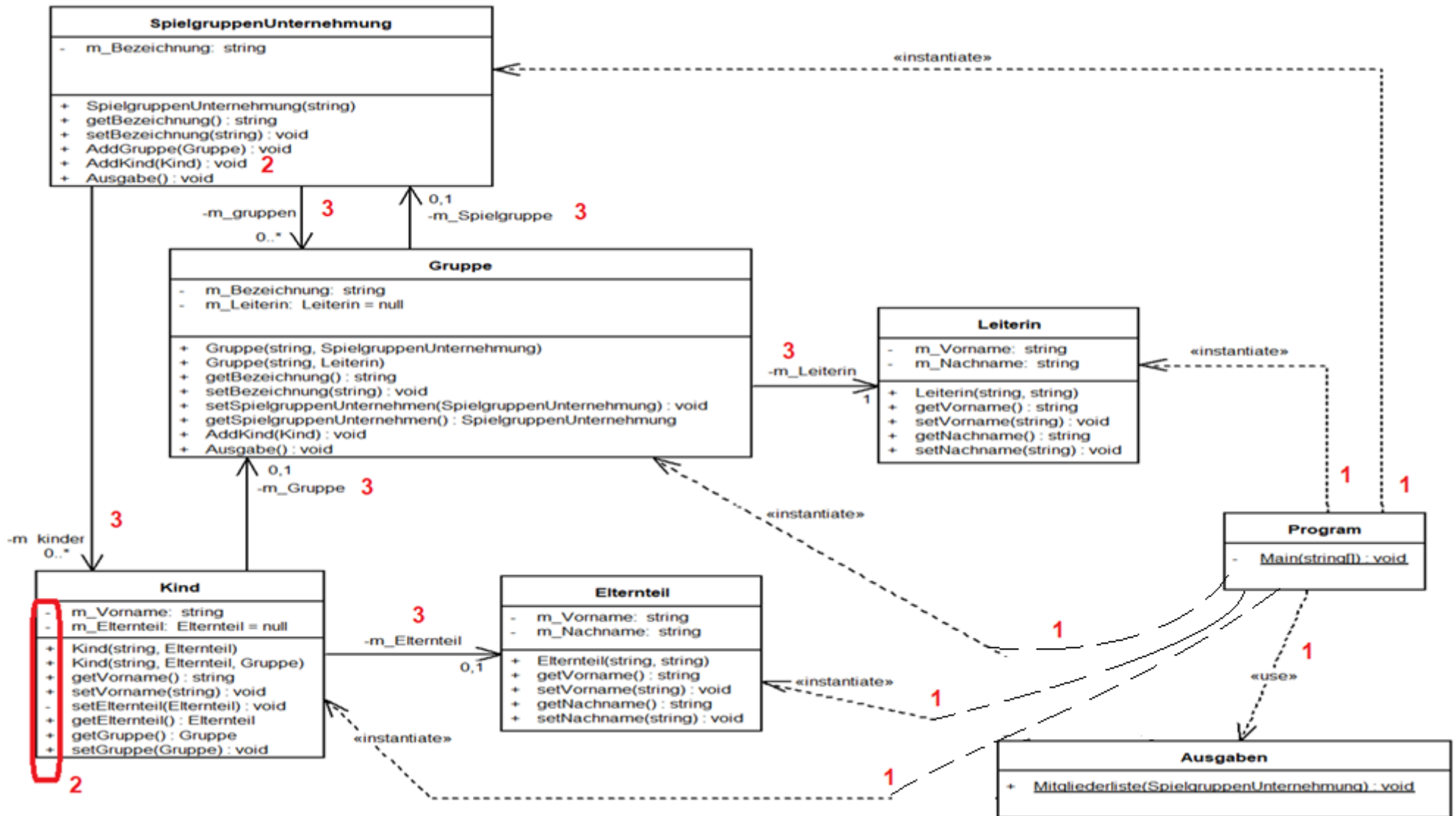
Welche Aussagen sind korrekt?

*//Tipp: direkt bedeutet: ohne über eine get-/set-Methode gehen zu müssen*

- ☐ In der Methode DoSomething() der Klasse R kann man direkt auf die Variable r4 lesend und schreibend zugreifen.
- ☒ In der Methode Dolt() der Klasse N kann man direkt auf die Variable m\_n2 lesend und schreibend zugreifen ☐ In der Methode DoSomething() der Klasse R kann man via Beziehungsvariable m\_n direkt auf die Variable m\_n1 lesend und schreibend zugreifen.
- ☐ In der Methode Dolt() der Klasse G kann man direkt auf die Variable m\_g1 zugreifen.
- ☒ In der Methode Dolt() der Klasse R kann man via Beziehungsvariable m\_n direkt auf die Variable m\_n2 lesend und schreibend zugreifen.
- ☐ In der Methode Dolt() der Klasse K kann man direkt auf die Variable m\_k2 zugreifen.
- ☒ In der Methode Dolt() der Klasse K kann man direkt, via Klassenname auf die Variable k3 zugreifen.
- ☐ In der Methode DoAnything() der Klasse R kann man direkt, via Klassenname, auf die Variable m\_d2 lesend und schreibend zugreifen.
- ☒ In der Methode Dolt() der Klasse D kann man direkt auf die Variable m\_d1 lesend und schreibend zugreifen.
- ☒ Innerhalb der Methode DoSomething() eines Objektes der Klasse R kann man ohne Instanziierung die Methode Dolt() des selben Objektes aufrufen
- ☐ Innerhalb der Methode DoAnything() der Klasse R kann man ohne Instanziierung auf das Datenfeld m\_n zugreifen.
- ☒ In der Methode Dolt() der Klasse A kann man direkt auf die Variable a3 lesend und schreibend zugreifen.
- ☐ In der Methode DoSomething() der Klasse D kann man direkt die Methode DoAnything() der selben Klasse aufrufen
- ☐ In der Methode DoSomething() der Klasse A kann man direkt auf die Variable m\_g1 zugreifen.
- ☒ Wenn man in der Dolt-Methode eines Objektes der Klasse D ein Objekt der Klasse A erstellen würde, könnte man via Referenz die DoAnything() Methode aufrufen.
- ☐ Wenn man in der Dolt-Methode eines Objektes der Klasse R ein Objekt der Klasse G erstellen würde, könnte man via Referenz die Dolt() Methode aufrufen.

## **2. Programmcode dokumentieren (28 P.)**

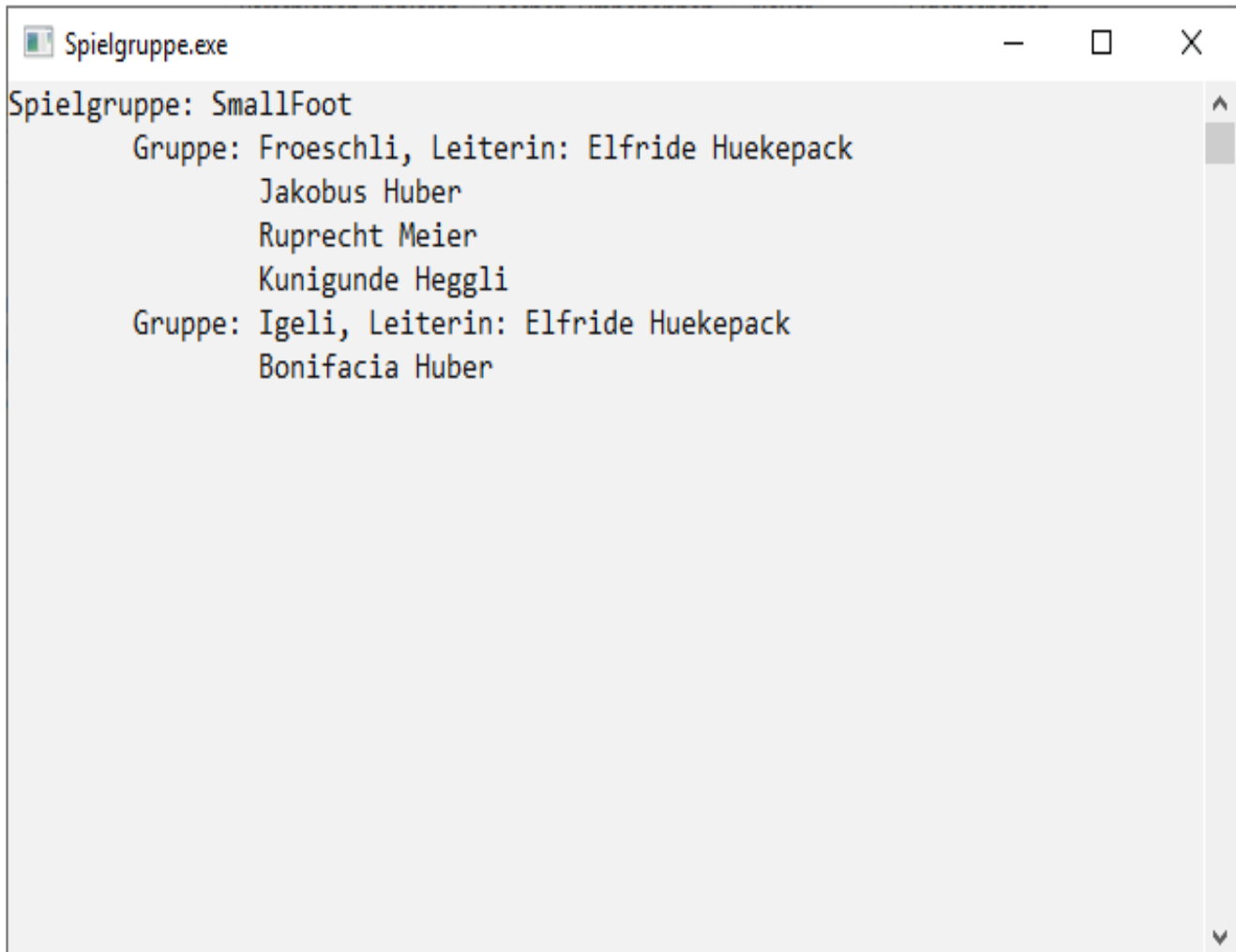
Im Anhang (Kapitel 4.2) finden Sie den kompletten Programmcode einer kleinen Applikation. Betrachten Sie den Programmcode bevor Sie mit den nachfolgenden Aufgaben weiterfahren. Vervollständigen Sie das untenstehende Klassendiagramm gemäss dem Programmcode. Stellen Sie auch alle Beziehungen zwischen den Klassen dar.



### 3. Programmcode interpretieren können (15 P.)

Was gibt die Applikation im Anhang auf dem Konsolenfenster aus?

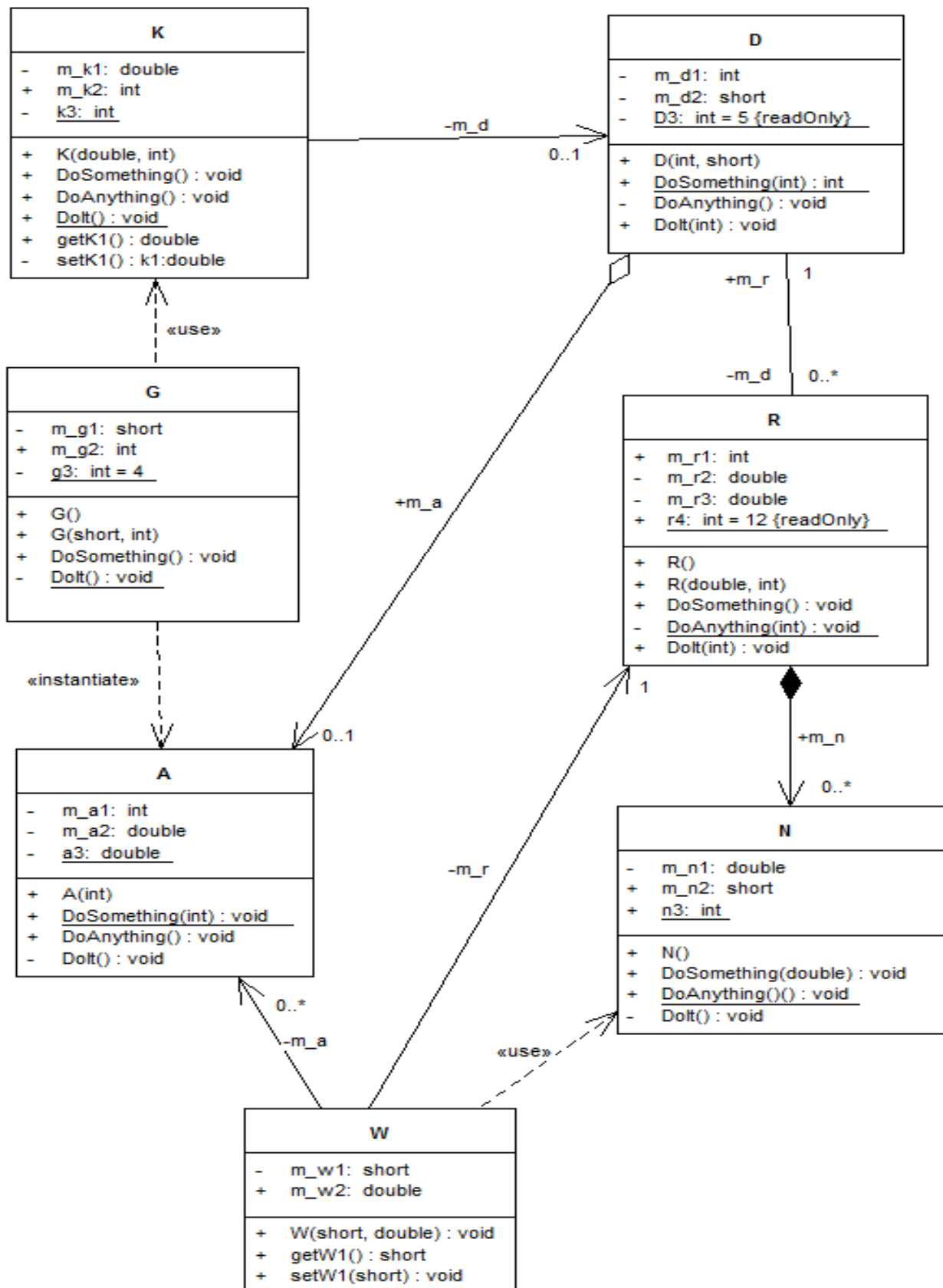
Notieren Sie Zeile für Zeile: [pro Zeile 1.5 P ausser bei GRP und Literin = 3P]



```
Spielgruppe.exe
Spielgruppe: SmallFoot
    Gruppe: Froeschli, Leiterin: Elfride Huekepack
        Jakobus Huber
        Ruprecht Meier
        Kunigunde Heggli
    Gruppe: Igeli, Leiterin: Elfride Huekepack
        Bonifacia Huber
```

## 4. Anhang

### 4.1 UML-Klassendiagramm zu den Aufgaben des Kapitels 1



## 4.2 Programmcode zur Aufgabe der Kapitel 2 und 3

Namen: .....

```
class Program {
    static void Main(string[] args) {
        SpielgruppenUnternehmung sf = new SpielgruppenUnternehmung("SmallFoot");
        Leiterin elf = new Leiterin("Elfride", "Huekepack");
        Gruppe fr = new Gruppe("Froeschli", elf);
        sf.AddGruppe(fr);
        Elternteil e = new Elternteil("Wigbert", "Huber");
        Kind k1 = new Kind("Jakobus", e);
        k1.setGruppe(fr);
        Kind k2 = new Kind("Ruprecht", new Elternteil("Osmunde", "Meier"));
        k2.setGruppe(fr);
        fr.AddKind(new Kind("Kunigunde", new Elternteil("Theodelind", "Heggli")));
        Gruppe ig = new Gruppe("Igeli", elf);
        ig.setSpielgruppenUnternehmen(sf);
        ig.AddKind(new Kind("Bonifacia", e));
        sf.AddGruppe(ig);
        Ausgaben.Mitgliederliste(sf);
        Console.ReadLine();
    }
}
```



M226\_P1 (2019).zip

```
public class SpielgruppenUnternehmung {
    private string m_Bezeichnung;
    private List<Gruppe> m_gruppen = new List<Gruppe>();
    private List<Kind> m_kinder = new List<Kind>();
    public SpielgruppenUnternehmung(string bezeichnung) {
        setBezeichnung(bezeichnung);
    }
    public string getBezeichnung() {
        return m_Bezeichnung;
    }
    public void setBezeichnung(string value) {
        m_Bezeichnung = value;
    }
    public void AddGruppe(Gruppe g) {
        if (m_gruppen.Contains(g) == false)
            m_gruppen.Add(g);
        if (g.getSpielgruppenUnternehmen() != this) //this = eigene Objektadresse
            g.setSpielgruppenUnternehmen(this);
    }
    public void AddKind(Kind k) {
        if (m_kinder.Contains(k) == false) //falls Kind nicht bereits in Array enthalten ist
            m_kinder.Add(k);
    }
    public void Ausgabe() {
        Console.WriteLine("Spielgruppe: " + getBezeichnung());
        foreach (Gruppe g in m_gruppen) {
            g.Ausgabe();
            foreach (Kind k in m_kinder) {
                if (k.getGruppe() == g) {
                    Console.WriteLine("\t\t" + k.getVorname() + " ");
                    if (k.getElternteil() != null)
                        Console.WriteLine(k.getElternteil().getNachname());
                }
            }
        }
    }
}
```

```
public class Leiterin {
    private string m_Vorname;
    private string m_Nachname;

    public Leiterin(string vorname, string nachname) {
        setVorname(vorname);
        setNachname(nachname);
    }

    public string getVorname() {
        return m_Vorname;
    }
    public void setVorname(string value) {
        m_Vorname = value;
    }
    public string getNachname() {
        return m_Nachname;
    }
    public void setNachname(string value) {
        m_Nachname = value;
    }
}

public class Gruppe {
    private string m_Bezeichnung;
    private Leiterin m_Leiterin = new Leiterin("Maximilia", "Mustermann");
    private SpielgruppenUnternehmung m_Spielgruppe = null;
    public Gruppe(string bezeichnung, SpielgruppenUnternehmung unternehmen) {
        setBezeichnung(bezeichnung);
        setSpielgruppenUnternehmen(unternehmen);
    }
    public Gruppe(string bezeichnung, Leiterin leit) {
        setBezeichnung(bezeichnung);
        if (leit != null)
            m_Leiterin = leit;
    }
    public string getBezeichnung() {
        return m_Bezeichnung;
    }
    public void setBezeichnung(string value) {
        m_Bezeichnung = value;
    }
    public void setSpielgruppenUnternehmen(SpielgruppenUnternehmung u) {
        if (m_Spielgruppe == null && u != null) {
            m_Spielgruppe = u;
            u.AddGruppe(this); //this = die eigene Adresse dieses Objektes
        }
    }
    public SpielgruppenUnternehmung getSpielgruppenUnternehmen() {
        return m_Spielgruppe;
    }
    public void AddKind(Kind k) {
        k.setGruppe(this);
        m_Spielgruppe.AddKind(k);
    }
    public void Ausgabe() {
        Console.WriteLine("\tGruppe: " + getBezeichnung() + ", ");
        Console.WriteLine("Leiterin: " + m_Leiterin.getVorname() + " " + m_Leiterin.getNachname());
    }
}
```



```
public class Kind {
    private string m_Vorname;
    private Elternteil m_Elternteil = null;
    private Gruppe m_Gruppe = null;
    public Kind(string vorname, Elternteil elter) {
        setVorname(vorname);
        setElternteil(elter);
    }
    public Kind(string vorname, Elternteil elter, Gruppe g) {
        setVorname(vorname);
        setElternteil(elter);
        setGruppe(g);
    }
    public string getVorname() {
        return m_Vorname;
    }
    public void setVorname(string value) {
        m_Vorname = value;
    }
    private void setElternteil(Elternteil value) {
        m_Elternteil = value;
    }
    public Elternteil getElternteil() {
        return m_Elternteil;
    }
    public Gruppe getGruppe() {
        return m_Gruppe;
    }
    public void setGruppe( Gruppe g) {
        if (g != m_Gruppe) {
            m_Gruppe = g;
            g.AddKind(this);
        }
    }
}

public class Elternteil {
    private string m_Vorname;
    private string m_Nachname;
    public Elternteil(string vorname, string nachname) {
        setVorname(vorname);
        setNachname(nachname);
    }
    public string getVorname() {
        return m_Vorname;
    }
    public void setVorname(string value) {
        m_Vorname = value;
    }
    public string getNachname() {
        return m_Nachname;
    }
    public void setNachname(string value) {
        m_Nachname = value;
    }
}

public class Ausgabe {
    public static void Mitgliederliste(SpielgruppenUnternehmung sg) {
        sg.Ausgabe();
    }
}
```