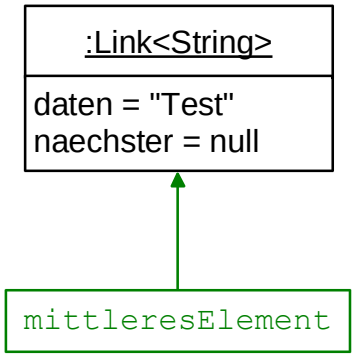
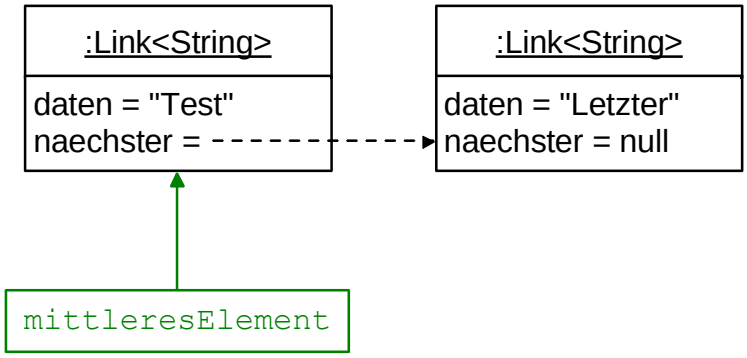
VL03, Lösung 1

```
public class LinkTest
{
    public static void main(String[] args)
    {
        // 1.
        Link<String> mittleresElement = new Link<String>("Test", null);

        // 2.
        mittleresElement.naechster = new Link<String>("Letzter", null);

        // 3.
        Link<String> anfang = new Link<String>("Erster", mittleresElement);
    }
}
```

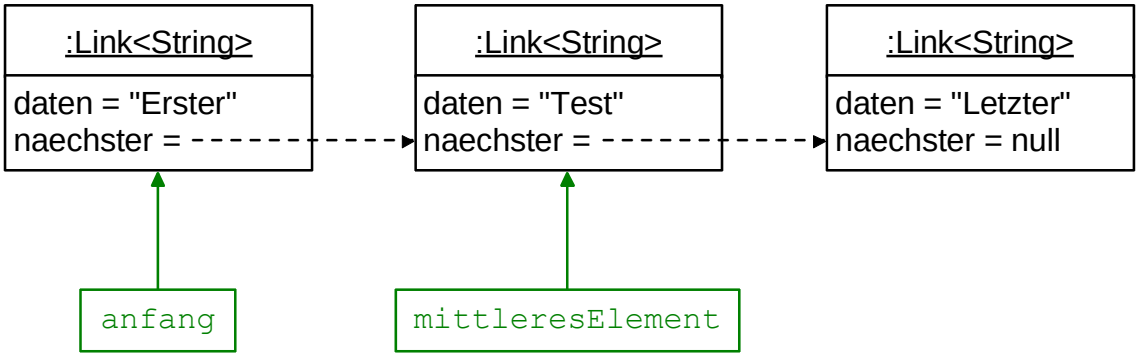
1)



2)



3)

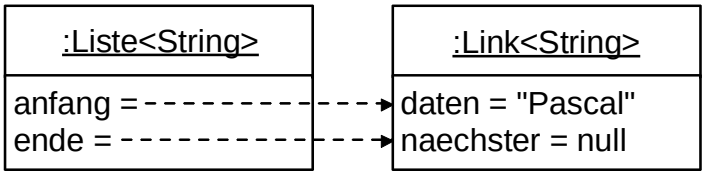


Alternativ können auch alle drei `Link`-Objekte in einer Programmzeile initalisiert werden, wenn es auf die Reihenfolge der einzelnen Schritte nicht ankommt:

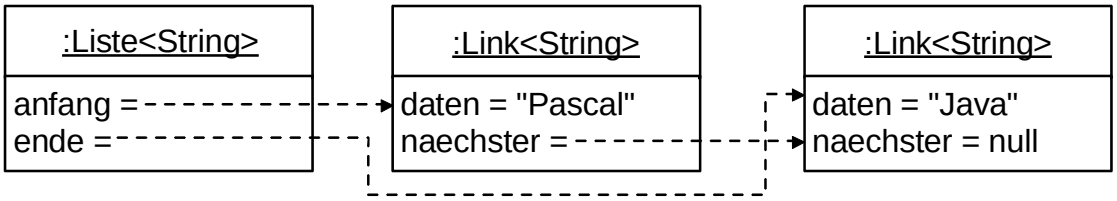
```
public class LinkTest
{
    public static void main(String[] args)
    {
        Link<String> anfang = new Link<String>("Erster",
            new Link<String>("Test",
            new Link<String>("Letzter", null)));
    }
}
```

VL03, Lösung 2

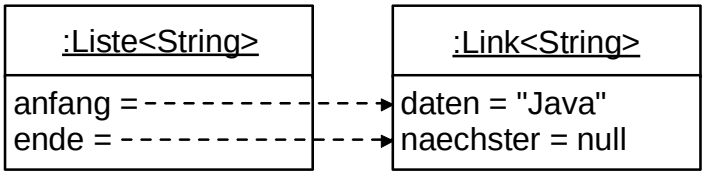
1)



2)



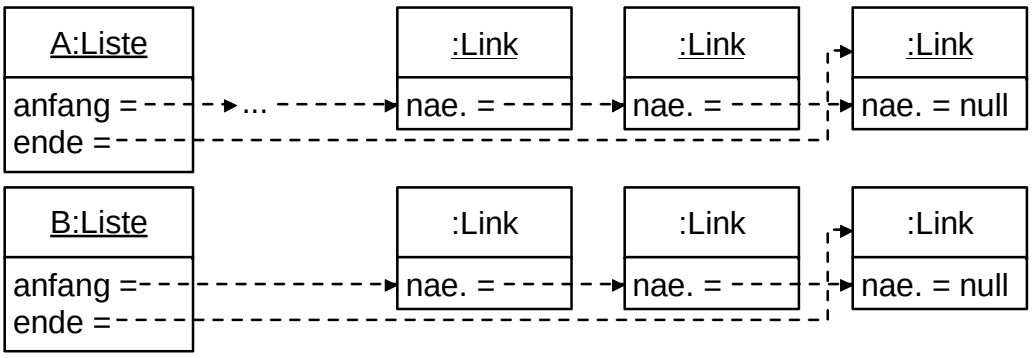
3)



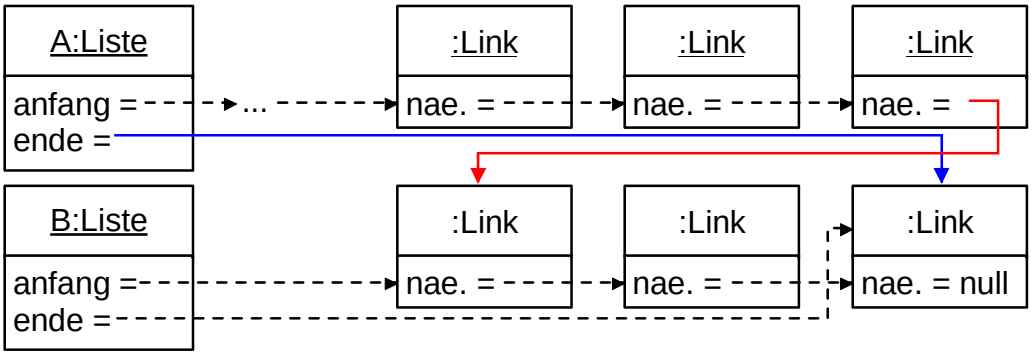
VL03, Lösung 3

In diesem Diagramm werden zur besseren Übersicht alle `daten` und Typparameter weggelassen, da diese hier keine Rolle spielen. Die Attribute, Referenzen und zugehörigen Objete sind natürlich dennoch weiterhin vorhanden, ebenso wie `Liste` und `Link` typisiert sind!

Vorher:

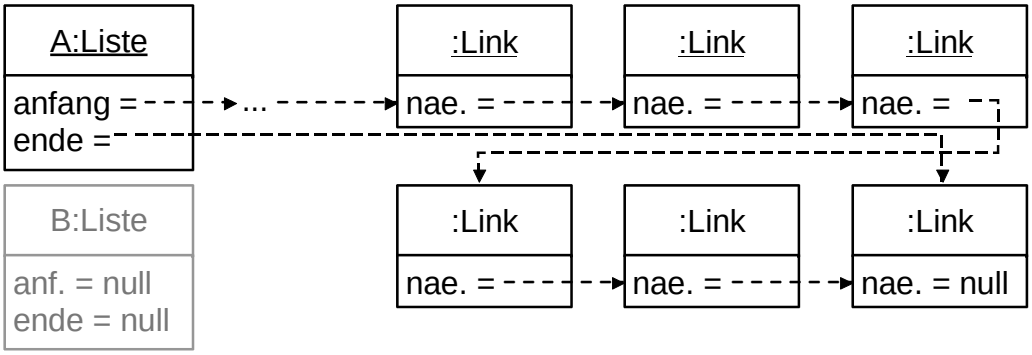


Nachher:



Das Verketten zweier Listen ist gefährlich, da weiterhin ein Zugriff über das zweite Listenobjekt `B` besteht. Wird hier z.B. das letzte Element gelöscht, so erfährt das kombinierte Listenobjekt `A` nichts davon, d.h. die dortige `ende`-Referenz zeigt auf ein Element, welches gar nicht mehr zur gemeinsamen Liste gehört.

Dieses Problem lässt sich beheben, indem nach der Verkettung das zweite Listenobjekt `B` von den verketteten `Link`-Objekten abgekoppelt wird, indem alle Refernzen auf `null` gesetzt werden:



VL03, Lösung 4

Die Textfarben entsprechen den Farben der Pfeile aus Lösung 3.

```
public boolean istLeer()
{
    return (anfang == null);
}

public void verketten(Liste<T> zweiteListe)
{
    assert(zweiteListe != null);

    // Verkettung nur durchführen, wenn beide Listen nicht identisch sind
    if (zweiteListe == this)
        return;

    // Eigentliche Verkettung
    if (istLeer())
    {
        // Sonderfall: die eigene Liste (this) ist leer!
        anfang = zweiteListe.anfang;
    }
    else
    {
        // Wenn die eigene Liste (this) nicht leer ist, muss
        // ende != null sein.
        assert(ende != null);

        ende.naechster = zweiteListe.anfang;
    }

    // Letztes Element der ersten Liste hat jetzt das erste Element der
    // zweiten Liste als Nachfolger. Dadurch kann nun das Listenend
    // auf das ende der zweiten Liste gesetzt werden.
    // Sonderfall: wenn die zweite Liste leer ist, darf das eigene ende
    // nicht überschrieben werden!
    if (zweiteListe.ende != null)
        ende = zweiteListe.ende;

    // Zweites Listenobjekt abkoppeln
    zweiteListe.anfang = zweiteListe.ende = null;
}
```

VL03, Lösung 5

```
public int entferneWerte(final T opfer)
{
    int anzGeloeschte = 0;

    Link<T> zeiger = anfang;
    Link<T> vorg = null;

    while (zeiger != null)
    {
        if (zeiger.daten.equals(opfer))
        {
            // Wenn am Anfang der Liste, anfang manipulieren
            if (vorg == null)
            {
                anfang = zeiger.naechster;
            }
            else
            {
                // Element aus Liste löschen; Vorgänger bleibt der alte
                vorg.naechster = zeiger.naechster;
            }

            anzGeloeschte++;
        }
        else
        {
            // Vorgänger aktualisieren
            vorg = zeiger;
        }

        // Zeiger aktualisieren
        zeiger = zeiger.naechster;
    }

    ende = vorg;

    return anzGeloeschte;
}
```