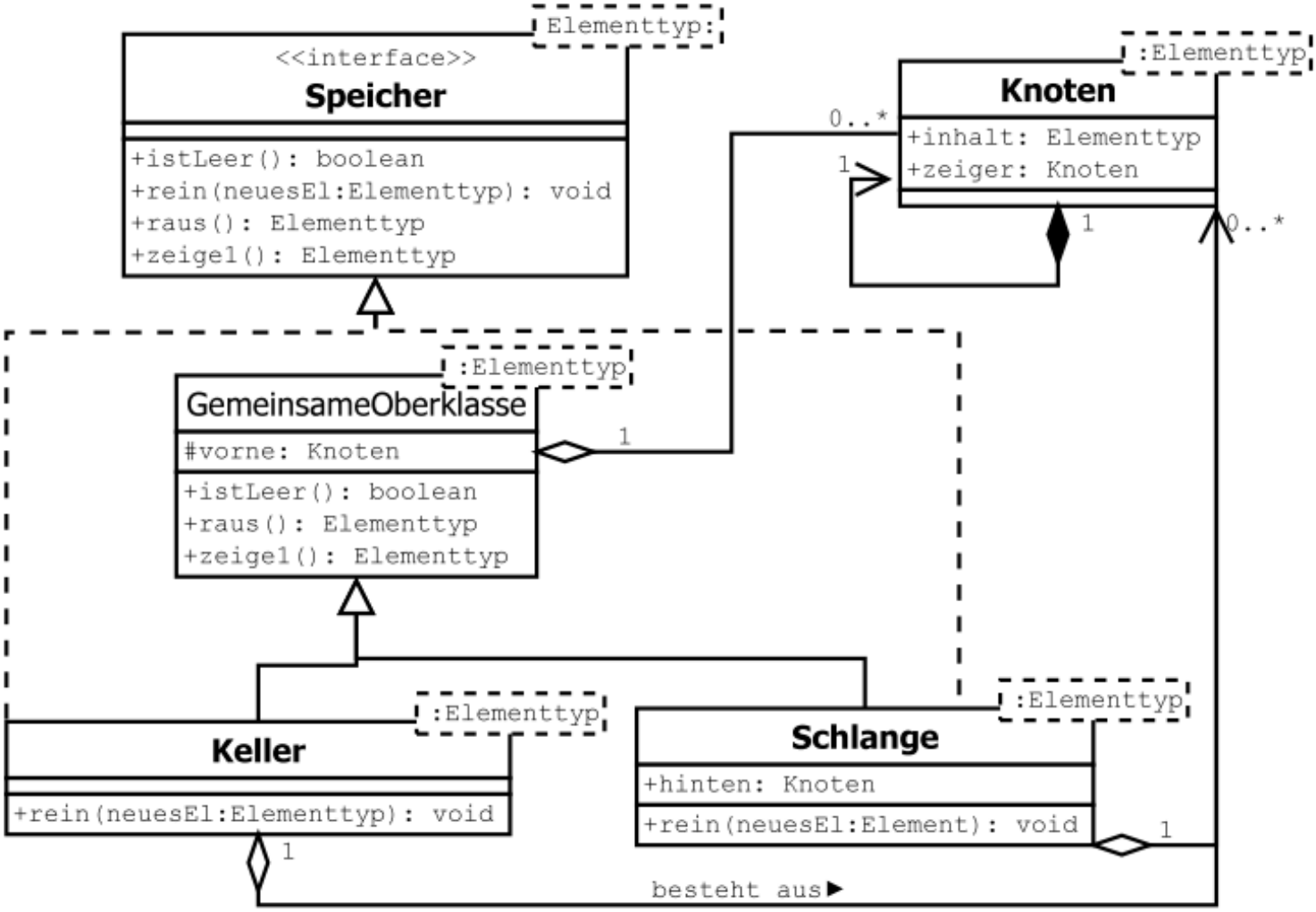
Keller und Schlange (mit UML-Diagramm)

Im Unterricht am Mi., 11.2.2015, war in Dia das UML-Diagramm für Keller und Schlange mit parametrisiertem Elementtyp erstellt worden:



Jetzt soll der Java-Text der einzelnen Klassen folgen. Der Quelltext wurde (noch) nicht kommentiert, dies soll als Hausaufgabe geschehen. Die abstrakten Datentypen waren im Unterricht bereits früher ausführlich behandelt, sodass der ausführliche Kommentar Anlass zur Wiederholung bietet und zeugt, ob alles verstanden und noch in Erinnerung ist.

1 Ein Interface als Versprechen, welche vier Methoden zur Bedienung jedes Speichers zur Verfügung gestellt werden müssen.
Der in spitzen Klammern übergebene *Elementtyp* ist ein Platzhalter, der später durch jeden beliebigen echten Typ ersetzt werden kann, sodass der Speicher dann nur Elemente vom angegebenen Typ aufnehmen kann.

```
public interface Speicher<Elementtyp>
{
  public void rein (Elementtyp neuesElement);

  public boolean istLeer();

  public Elementtyp zeige1();

  public Elementtyp raus();
}
```

```
public class Knoten<Elementtyp>
{
  Elementtyp inhalt;
  Knoten zeiger;
}
```

2 Für die dynamische Implementation gibt es den Knoten, der mit seinem Zeiger wieder auf einen Knoten zeigen kann. Die Attribute sind hier ohne Sichtbarkeits-Kennzeichnung (default), sodass ein Konstruktor oder *get-/set*-Methoden überflüssig sind.

3 Weil Keller und Schlange drei völlig identische Methoden haben, werden diese in eine gemeinsame Oberklasse ausgelagert und von dort geerbt. Von der Oberklasse selbst soll kein Objekt erzeugt werden dürfen, weshalb sie als „abstrakt“ definiert wird.
Das als „protected“ definierte Attribut *vorne* ist nur innerhalb der Klasse selbst und ihren Erben (Spezialisierungen, Unterklassen) sichtbar und verwendbar, hier also nur in *GemeinsameOberklasse, Keller* und *Schlange*.

```
public abstract class GemeinsameOberklasse<Elementtyp>
{
  protected Knoten<Elementtyp> vorne = null;

  public boolean istLeer()
  {
    return (vorne==null);
  }

  public Elementtyp zeige1()
  {
    return (vorne.inhalt);
  }

  public Elementtyp raus()
  {
    Elementtyp hilf = zeige1();
    vorne = vorne.zeiger;
    return (hilf);
  }
}
```

4 Der Keller erbt von der Oberklasse und da er auch noch eine *rein*-Methode enthält, verwirklicht er so insgesamt das Versprechen des Interfaces *Speicher*.

```
public class Keller<Elementtyp>  extends GemeinsameOberklasse<Elementtyp>  implements Speicher<Elementtyp>
{
  public void rein (Elementtyp neuesElement)
  {
    Knoten<Elementtyp> neuerKn = new Knoten<Elementtyp>();
    neuerKn.inhalt = neuesElement;
    neuerKn.zeiger = vorne;
    vorne = neuerKn;
  }
}
```

5 Die Schlange erbt ebenfalls von der Oberklasse, fügt als weiteres Attribut *hinten* und ein eigenes *rein* hinzu

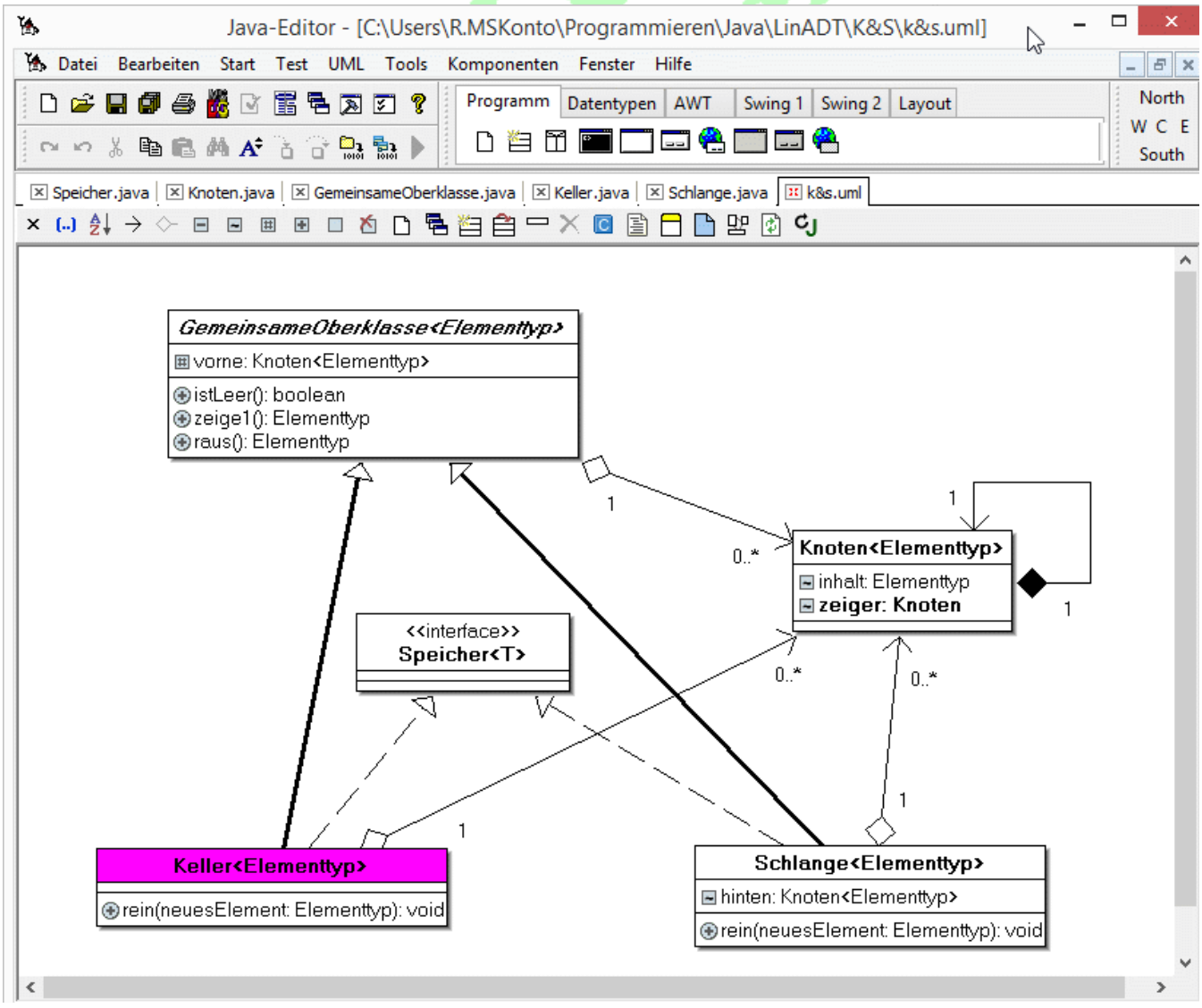
```
public class Schlange<Elementtyp> extends GemeinsameOberklasse<Elementtyp> implements Speicher<Elementtyp>
{
  private Knoten<Elementtyp> hinten = null;

  public void rein (Elementtyp neuesElement)
  {
    Knoten<Elementtyp> neuerKn = new Knoten<Elementtyp>();
    neuerKn.inhalt = neuesElement;
    neuerKn.zeiger = null;
```

(Forts. s. nächste Seite)

Fortsetzung der Schlange:

```
    if (istLeer())
    {
      vorne = neuerKn;
    }
    else
    {
      hinten.zeiger = neuerKn;
    }
    hinten = neuerKn;
  }
}
```

Mit dem Java-Editor kann aus diesen Dateien automatisch ein UML-Klassendiagramm erzeugt werden. Während die Vererbungspfeile auch automatisch entstehen, müssen die Aggregations- und Kompositions-Verbindungen von Hand hinzugefügt und mit Multiplizitäten versehen werden. Der Javaeditor kennzeichnet den parametrisierten Elementtyp nicht in einem eigenen Rahmen; der Name der abstrakten Klasse wird – anders als bei Dia (dort dünne Schrift) – hier kursiv dargestellt:



6 Um zu zeigen, dass die programmierten Speicher auch wunschgemäß funktionieren, wurde eine Testklasse geschrieben und ausgeführt. Erkläre die Konstruktionen sowie die gezeigte Ausgabe!

```
public class Test
{
 public static void main (String[] s)
 {
  System.out.println(" Zunächst wird ein Keller für Ganzzahlen erzeugt und getestet:");
  Keller<Integer> ki = new Keller<Integer>();
  System.out.println(" - anfangs ist der Keller leer: "+ki.istLeer());
  System.out.print(" - jetzt kommen nacheinander in den Keller:");
  ki.rein(1); System.out.print(" "+ki.zeige1());
  ki.rein(2); System.out.print(" "+ki.zeige1());
  ki.rein(3); System.out.println(" "+ki.zeige1());
  System.out.println(" - der Keller liefert jetzt istLeer = "+ki.istLeer());
  System.out.print(" - und jetzt kommt alles nacheinander raus:");
  while (!ki.istLeer())
  {
   System.out.print(" "+ki.raus());
  }
  System.out.println(" Fertig!");
  System.out.println();

  System.out.println(" Nun wird ein Keller für Strings erzeugt und getestet:");
  Keller<String> ks = new Keller<String>();
  System.out.println(" - anfangs ist der Keller leer: "+ks.istLeer());
  System.out.print(" - jetzt kommen nacheinander in den Keller:");
  ks.rein("eins"); System.out.print(" "+ks.zeige1());
  ks.rein("zwei"); System.out.print(" "+ks.zeige1());
  ks.rein("drei"); System.out.println(" "+ks.zeige1());
  System.out.println(" - der Keller liefert jetzt istLeer = "+ks.istLeer());
  System.out.print(" - und jetzt kommt alles nacheinander raus:");
  while (!ks.istLeer())
  {
   System.out.print(" "+ks.raus());
  }
  System.out.println(" Fertig!");
  System.out.println();

  System.out.println(" Zum Schluss folgt eine Schlange für Kommazahlen:");
  Schlange<Double> sd = new Schlange<Double>();
  System.out.println(" - anfangs ist die Schlange leer: "+sd.istLeer());
  System.out.println(" - jetzt kommen nacheinander in die Schlange: 1.1 2.2 3.3");
  sd.rein(1.1);
  sd.rein(2.2);
  sd.rein(3.3);
  System.out.println(" - dabei steht in der Schlange vorne immer "+sd.zeige1());
  System.out.println(" - die Schlange liefert jetzt istLeer = "+sd.istLeer());
  System.out.print(" - und jetzt kommt alles nacheinander raus:");
  while (!sd.istLeer())
  {
   System.out.print(" "+sd.raus());
  }
  System.out.println(" Fertig!");
  System.out.println();
 }
}
```

```
Zunächst wird ein Keller für Ganzzahlen erzeugt und getestet:
- anfangs ist der Keller leer: true
- jetzt kommen nacheinander in den Keller: 1 2 3
- der Keller liefert jetzt istLeer = false
- und jetzt kommt alles nacheinander raus: 3 2 1 Fertig!

Nun wird ein Keller für Strings erzeugt und getestet:
- anfangs ist der Keller leer: true
- jetzt kommen nacheinander in den Keller: eins zwei drei
- der Keller liefert jetzt istLeer = false
- und jetzt kommt alles nacheinander raus: drei zwei eins Fertig!

Zum Schluss folgt eine Schlange für Kommazahlen:
- anfangs ist die Schlange leer: true
- jetzt kommen nacheinander in die Schlange: 1.1 2.2 3.3
- dabei steht in der Schlange vorne immer 1.1
- die Schlange liefert jetzt istLeer = false
- und jetzt kommt alles nacheinander raus: 1.1 2.2 3.3 Fertig!
```