Aufgabe 1

Wähle den ersten Kontakt von aListe aus.
Wiederhole, solange ein Kontakt von aListe ausgewählt ist:
    falls der zu löschende Nachname dem Namen des ausgewählten Kontakts entspricht,
        entferne den ausgewählten Kontakt aus aListe
        breche die Methode ab und gib true zurück
    wähle den nächsten Kontakt von aListe aus.

Aufgabe 2

a)
Wähle den ersten Kontakt von aListe aus.
Wiederhole, solange ein Kontakt von aListe ausgewählt ist:
    Falls der Nachname des einzufügenden Kontakts alphabetisch vor
    dem Nachnamen des ausgewählten Kontakts liegt:
        Füge den Kontakt in aListe ein.
        Breche die Schleife ab.
    Wähle das nächste Element von aListe aus.
Falls kein Element von aListe ausgewählt ist:
    Füge den Kontakt am Ende der Liste ein.

```
public void einfügenAlph(Kontakt pk)
{
   Kontakt kAkt;
   aListe.toFirst();
   while (aListe.hasAccess())
   {
      kAkt = aListe.getContent();
      if (pk.getNachname().compareTo(kAkt.getNachname()) < 0)
      {
         aListe.insert(pk);
         break;
      }
      aListe.next();
   }
   if (!aListe.hasAccess())
   {
      aListe.append(pk);
   }
}
```

b)
Wähle den ersten Kontakt von aListe aus.
Wiederhole, solange ein Kontakt von aListe ausgewählt ist:
    Falls der gesuchte Nachname alphabetisch vor dem Nachnamen des ausgew. Kontakts liegt:
        Breche die Methode ab und gib null zurück.
    Falls der gesuchte Nachname dem Nachnamen des ausgewählten Kontakts entspricht:
        Breche die Methode ab und gib den ausgewählten Kontakt zurück.
    Wähle das nächste Element von aListe aus.
Gib null zurück.

```
public Kontakt suchen(String pName)
{
   Kontakt kAkt;

   aListe.toFirst();
   while (aListe.hasAccess())
   {
      kAkt = aListe.getContent();
      if (pName.compareTo(kAkt.getNachname()) < 0)
      {
         return null;
      }
      if (pName.equals(kAkt.getNachname()))
      {
         return kAkt;
      }
      aListe.next();
   }
   return null;
}
```

c)
Wenn die Liste des Adressbuchs für andere Klassen öffentlich wäre, könnten diese Klassen mithilfe der Methoden der Klasse List an beliebiger Stelle Kontakte einfügen. Das könnte die alphabetische Reihenfolge durcheinanderbringen. Eine Folge wäre, dass weder das alphabetisch sortierte Einfügen noch die Suchfunktion korrekt funktionieren würden, da diese abbrechen, sobald der gesuchte Name alphabetisch VOR dem aktuellen Namen der Liste liegt.
Aus diesem Grund sollte die Liste nur von Methoden der Klasse Adressbuch verändert werden.

Aufgabe 3

a)

0.	Picasso	Dalí	Kandinsky	Cézanne	Monet
1.	Cézanne	Picasso	Dalí	Kandinsky	Monet
2.	Cézanne	Dalí	Picasso	Kandinsky	Monet
3.	Cézanne	Dalí	Kandinsky	Picasso	Monet
4.	Cézanne	Dalí	Kandinsky	Monet	Picasso

b)
Der Algorithmus sortiert die Liste alphabetisch nach den Nachnahmen.
Es wird der Selection Sort angewendet (der jeweils das Minimum in der verbleibenden Liste sucht).

c)
Die Variable a zählt mit, wie viele Elemente am Anfang der Liste schon richtig einsortiert sind.
Sie wird dann beim Einfügen des nächsten Minimums benutzt, um die Liste an die richtige Stelle „vorzuspulen“.

d)

```
public void sortieren()
{
   int i, anfang;
   Kontakt kMin, kAkt;

   // anfang gibt an, wie viele Elemente schon richtig einsortiert sind
   anfang = 0;

   // Die äußere Schleife wiederholt die Suche nach dem Minimum
   aListe.toFirst();
   while (aListe.hasAccess())
   {
      // Die Suche beginnt nach den schon sortierten Kontakten.
      // Am Ende der äußeren Schleife wird bis dort "vorgespult".
      // Der Kontakt an dieser Stelle wird als Minimum angenommen.
      kMin = aListe.getContent();

      // Die restliche Liste wird nach dem Minimum durchsucht.
      aListe.next();
      while (aListe.hasAccess())
      {
         kAkt = aListe.getContent();
         if (kAkt.getNachname().compareTo(kMin.getNachname()) < 0)
         {
            kMin = kAkt;
         }
         aListe.next();
      }

      // Das Minimum wird erneut gesucht und aus der Liste entfernt.
      aListe.toFirst();
      while (kMin != aListe.getContent())
      {
         aListe.next();
      }
      aListe.remove();

      // Füge das Minimum an der richtigen Position ein.
      aListe.toFirst();
      for (i = 0; i < anfang; i++)
      {
         aListe.next();
      }
      if (aListe.hasAccess())
      {
         aListe.insert(kMin);
      }
      else
      {
         aListe.append(kMin);
      }
      // erhöhe die Anzahl der richtig einsortierten Elemente um 1
      anfang++;
   }
}
```