Verwenden Sie für die unten stehenden Aufgaben folgende Definition des Interface Folge<T>:

```
import java.util.NoSuchElementException;

public interface Folge<T> extends Iterable<T> {

        int size();
        boolean isEmpty();
        boolean contains(T e);
        T get(int pos) throws IndexOutOfBoundsException;
        void set(int pos, T e) throws IndexOutOfBoundsException;
        void insert(int pos, T e) throws IndexOutOfBoundsException;
        void remove(int pos) throws IndexOutOfBoundsException;
        void delete(T e);
}
```

Die Positionsnummerierung für die Methoden insert(), {get(), set(), remove() beginnt bei 0.
Die Methoden get(), set() und remove() lösen jeweils eine Exception aus, wenn $pos < 0$ oder $pos \geq size$.
Für die Methode insert() ist $pos == size$ zulässig; die Methode entspricht dann der Methode add(), die ein Element *hinten* anfügt.

Aufgabe 6.1
Implementieren Sie in einer Klasse FolgeAlsDynArray<T> das Interface Folge<T> mittels eines dynamischen Arrays.

Aufgabe 6.2
Implementieren Sie eine Klasse DVL<T>, die die Datenstruktur einer „doppelt verkettete Liste“ implementiert (oder nutzen Sie Ihre Lösung aus A 4.4).
Implementieren Sie in einer Klasse FolgeDVL<T> das Interface Folge<T> mittels einer DVL.

Die DVL soll mindestens über folgende Methoden verfügen:

int size(), **boolean** isEmpty(), **boolean** contains(T v), T getFirst(), T getLast
void insert(T v), **void** append(T v) (vorne einfügen bzw hinten anhängen)
void removeFirst(), **void** removeLast(), **void** delete(T value)

Aufgabe 6.3
Implementieren Sie eine Klasse FolgeUtil, die die folgenden (*statischen, generischen*) Methoden zum Umgang mit Folgen zur Verfügung stellt:

- **int** frequency(Folge<T> f, T v) liefert die Anzahl der Element in f, die „gleich“ v sind
  (oder anders ausgedrückt: liefert die Häufigkeit, mit der das Element v in f vorkommt)
- **void** swap(Folge<T> f, **int** i, **int** j) vertauscht die Einträge an den Positionen i und j der Folge f
- **void** rotate(Folge<T> f, **int** d): rotiert die Folge um d-viele Elemente
  Beispiel: die Folge $f = [10, 42, 0, 8, 15]$ wird durch rotate(f, 2) zu $[8, 15, 10, 42, 0]$

Aufgabe 6.4

a) Schreiben Sie eine Klasse Song, die Musikstücke darstellt: Ein song hat

- einen Titel
- einen Interpreten
- eine Länge (nutzen Sie hierfür als Datentyp die Klasse java.time.Duration)
- ein Erscheinungsjahr
- einen Songtext

Die Daten für Titel, Interpret, Länge und Erscheinungsjahr sollen dem Konstruktor übergeben werden, der Song-Text kann durch eine set-Methode gesetzt werden. Für all diese Daten soll es entsprechende get-Methoden geben.
Die toString-Methode soll <Nummer>, <Interpret>, <Titel>, <Dauer> des songs liefern.
equals(Object o) prüft die Gleichheit zweier Songs anhand von Titel und Erscheinungsjahr.

b) Implementieren Sie eine Klasse Playlist.
Eine Playlist enthält eine Folge von Songs.
Implementieren Sie folgende Methoden für Playlist:

- **void** add(Song song): fügt einen song der playlist hinzu
- **void** delete(Song song): löscht einen song aus der playlist
- **void** play(): spielt die songs der playlist in der gespeicherten Reihenfolge ab
- **void** play(Song song): spielt (nur) den angegeben Song ab
- **void** shuffle(): spielt die playlist in zufälliger (gemischter) Reihenfolge ab (ohne die playlist selber zu ändern)

Bemerkungen:

- Das „Abspielen“ eines Songs simulieren wir durch Ausgabe des Songtexts.
- Für shuffle bietet es sich an, einen Zufallszahlengenerator der Klasse Random zu benutzen.

(*) Zusatzaufgabe: wie kann man verhindern, dass „zufällig“ ein Song häufiger als andere oder mehrmals hintereinander abgespielt wird?
Implementieren Sie einen „RandomIterator“, der die Elemente einer Folge in zufälliger Reihenfolge, aber ohne Wiederholungen liefert.