Contents

1 Datenstrukturen

1.1 ArrayList

Die ArrayList in Java ist eine dynamisch wachsende Liste, die auf dem zugrunde liegenden Array basiert. Sie erweitert die Funktionalität von Arrays durch automatische Größenanpassung und bietet eine flexible Möglichkeit, Elemente hinzuzufügen, zu löschen und darauf zuzugreifen.

Anwendungsfälle:
- Speicherung von Elementen einer Liste
- Zugriff auf Elemente durch Index
- Wenn die Größe der Liste nicht im Voraus bekannt ist
- Wenn häufiger Einfüge- und Löschoperationen erforderlich sind

Syntax:

```java
// Import der ArrayList-Klasse
import java.util.ArrayList;

// Deklaration und Initialisierung
ArrayList<Integer> myArrayList = new ArrayList<>();

// Elemente hinzufuegen
myArrayList.add(1);

// Zugriff auf ein Element
int element = myArrayList.get(0);

// Iteration durch die ArrayList
for (int i = 0; i < myArrayList.size(); i++) {
    System.out.println(myArrayList.get(i));
}
```

Funktionalität unter der Haube:
- Intern durch Array verwaltet, welches ggf. neu dimensioniert werden muss
- Neudimensionierung führt zu O(n), da neues Array und alle Elemente kopiert werden. Wird allerdings selten gemacht und dann i.d.R. direkt verdoppelt, somit amortisiert O(1)

Laufzeitkomplexität

Operation	Add	Remove	Get	Contains	Next	Size
Time Complexity	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(1)$

1.2 LinkedList

Dynamische Datenstruktur bestehend aus Knoten, die jeweils auf den nächsten Knoten in der Sequenz zeigen Zwei Haupttypen: Singly Linked (Zeigt nur auf Nachfolger), Doubly Linked (Zeigt auf Vor- und Nachfolger).

Anwendungsfälle:
- Bei häufigen Einfüge- und Löschoperationen an verschiedenen Positionen
- Wenn die Größe der Liste variabel ist

Syntax:

```java
// Import der ArrayList-Klasse
import java.util.ArrayList;

// Deklaration und Initialisierung
ArrayList<Integer> myArrayList = new ArrayList<>();

// Elemente hinzufuegen
myArrayList.add(1);

// Zugriff auf ein Element
int element = myArrayList.get(0);

// Iteration durch die ArrayList
for (int i = 0; i < myArrayList.size(); i++) {
    System.out.println(myArrayList.get(i));
}
```

Funktionalität unter der Haube:
- Jeder Knoten erhält Verweise auf Nach- (ggf. Vorgänger)
- Einfügen und Löschen an am Ende/Anfang konstant, ansonsten linear, da update der Verweise

Laufzeitkomplexität

Operation	Add	Remove	Get	Contains	Next	Size
Time Complexity	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$

1.3 Stack

Agiert nach **L**ast-**I**n-**F**irst-**O**ut (**LIFO**) Prinzip: Das letzte hinzugefügte Element wird als Erstes entfernt. Unterstützt lediglich Hinzufügen (Push) und Entfernen (Pop).

Anwendungsfälle:
- Rückverfolgung von Funktionsaufrufen (Call Stack Aufbau bspw.)
- Umkehrung von Zeichenketten/Ausdrücken
- Undo-Mechanismen

Syntax:

```java
// Verwendung des Stack-Datentyps
import java.util.Stack;

// Deklaration und Initialisierung
Stack<Integer> myStack = new Stack<>();

// Elemente hinzuf gen
myStack.push(1);

// Element entfernen
int poppedElement = myStack.pop();
```

Funktionalität unter der Haube:
- Ein Stack kann durch ein Array oder eine verkettete Liste implementiert werden
- Die Elemente werden auf dem Stack oben hinzugefügt und entfernt
- Schnelle Hinzufüge- und Entfernungsoperationen (konstante Laufzeit O(1))
- Keine Suchmöglichkeit, ggf. aber Möglichkeit oberstes Element anzuschauen (peek) - Custom Suche implementierbar, dann O(n) wahrscheinlich

Laufzeitkomplexität

Operation	Push	Pop
Time Complexity	$O(1)$	$O(1)$