**Nachdenkzettel Collections:**

**Aufgabe 1:**

Die Wahl zwischen ArrayList und LinkedList hängt von den spezifischen Anforderungen und dem Anwendungsfall ab:

ArrayList:

* Besser geeignet für den Zugriff auf Elemente über den Index.
* Effizienter bei Lesezugriffen und Iterationen.
* Gute Wahl, wenn Elemente häufig gelesen, aber selten gelöscht oder eingefügt werden.

LinkedList:

* Besser geeignet für häufige Einfüge- und Löschoperationen, insbesondere am Anfang oder in der Mitte der Liste.
* Overhead durch zusätzliche Verkettungsstruktur.
* Gute Wahl, wenn die Reihenfolge wichtig ist und dynamisch angepasst wird.

**Aufgabe 2:**

Es gibt deutliche Effizienzunterschiede zwischen den verschiedenen Collection-Arten.

• LinkedList: Bei allen Aktionen, außer "retain all", langsamer oder ähnlich schnell wie ArrayList.

• CopyOnWriteArrayList: Ineffizient bei Datenänderungen.

• HashMap und ArrayList: Vergleichsweise schnelle Leistung.

**Aufgabe 3:**

Die CopyOnWriteArrayList ist nicht unbedingt langsam, sondern wurde für bestimmte Anwendungsfälle optimiert. Diese Klasse erstellt eine Kopie der Listendaten, wenn Änderungen vorgenommen werden, um die Konsistenz während der Iterationen zu gewährleisten. Daher kann sie im Vergleich zu anderen Listenimplementierungen weniger effizient erscheinen, wenn die Liste häufig geändert wird. Sie eignet sich jedoch gut für Situationen, in denen Iterationen häufiger vorkommen als Änderungen.

**Aufgabe 4:**

Der angegebene Code versucht, alle Elemente einer Liste zu entfernen, die größer als 5 sind. Dies führt zu einem ConcurrentModificationException-Fehler, da die Liste während der Iteration geändert wird.

Warum passiert das?

Das remove() innerhalb der Iteration verändert die Struktur der Liste, was zu Inkonsistenzen führt.

Wie kann dieser Effekt umgangen werden?

Eine mögliche Lösung ist die Verwendung eines Iterator-Objekts und der remove()-Methode des Iterators, um Elemente sicher zu entfernen:

Iterator<Integer> itr = list.iterator();

while (itr.hasNext()) {

int i = itr.next();

if (i > 5) {

itr.remove();

}

}

Dies ermöglicht das sichere Entfernen von Elementen während der Iteration, ohne eine ConcurrentModificationException auszulösen.