Nachdenkzettel: Collections

1.

ArrayList:
Besteht aus einem dynamisch skalierbaren Array und kann schnell mittels eines Indexes auf Inhalte in der Liste zugreifen. Im durschnitt hat man beim Hinzufügen eine O(1) Komplexität, da man nur ans Ende etwas hinzufügt. Der Worst-Case ist O(n), da der Array voll ist und man einen neuen Größeren erstellen muss.

LinkedList:

Besteht aus einer doppelten verketteten Liste und zeichnet sich durch das schnelle Hinzufügen oder Entfernen von Elementen in der Mitte aus. Der Grund dafür ist das nicht verschieben der Elemente, wie die ArrayList, welche die Indizes anpassen muss.
Das Einfügen und Zugreifen bei der LinkedList liegt bei einer Komplexität von O(n).

2.

LinkedList, CopyOnWriteArrayList, FastList, TreeList:

Gut für das Hinzufügen von unterschiedlichen Elementen an einem bestimmten Index

CopyOnWriteArrayList:

Gut für das Hinzufügen von 1000 Mal 1000 Elemente

CopyOnWriteArrayList:

Gut für das Hinzufügen von 1000 Mal 1000 Elemente an einem bestimmten Index

LinkedList:

Güt für das Löschen aller Einträge in einer Liste

TreeList:

Gut bei einem Check, ob Elemente in der Collection, welche 1000 groß ist, enthalten sind

Gut einem Check, ob Elemente in einer Collection, welche 5000

FastList, LinkedList:

Gut einem get von einer Größe von 50000

TreeList:

Gut für einen Command indexOf, bei 5000 Mal

TreeList:

Gut bei einer Iteration von 100000 Mal in einer Collection

TreeList:

Gut bei einem LastIndexOf bei 1000

TreeList:

Gut bei ListIterator von 100000 Mal

FastList:

Gut bei ListIterator bei einem gebebenen Index von 100000 Mal

TreeList, CopyOnWriteArrayList, FastList:

Gut beim löschen von Einträgen in einer 10000 Liste

CopyOnWriteArrayList:

Gut beim löschen von Einträgen mit einem gegebenen Index

WriteArrayList:

Gut beim Löschen von allen Einträgen (1000 Elemente)

ArrayList, Vector,FastList:

Gut bei retainAll von 10 Mal

CopyOnWriteArrayList, FastList:

Gut beim bearbeiten von einzelnen Einträgen in einer Liste

FastList:

Gut beim Erstellen von views einer Collection für 25000 Elemente auf einer 100000 Liste

TreeList, CopOnWriteArrayList:

Gut für das umwandeln in einen Array der Größe 5000

3.

Die CopyOnWriteArrayList erstellt nach jedem add oder remove ein neues Array mit den jeweiligen neuen oder alten Werten. Da man bei sehr großen Arrays dies besonders stark merkt, scheint diese Art von List langsam.

4.

Man bedient sich hier an einem Iterator (Pointer), welcher wie bei einer For-Schleife durch die Liste iteriert und schaut, ob ein nächster Wert in der Liste ist und wenn ja diesen abhackt/durchgeht. Hier ist die Besonderheit, dass das Durchgehen mit einer Variablen i gespeichert wird und ab einer 6 alle Werte aus der Liste löscht. Die Besonderheit ist jedoch falsch da wenn man Gebrauch von einem Iterator macht, man die Liste nicht währenddessen ändern darf, es sei denn, man tut dies über den Iterator selbst.