Java Collections Cheat Sheet

Java Collections Cheat Sheet

Java Concept Of The Day

Basics

What is Java Collection Framework?

Java Collection Framework is a framework which provides some predefined classes and interfaces to store and manipulate the group of objects. Using Java collection framework, you can store the objects as a List or as a Set or as a Queue or as a Map and perform basic operations like adding, removing, updating, sorting, searching etc.... with ease.

Why Java Collection Framework?

Earlier, arrays are used to store the group of objects. But, arrays are of fixed size. You can't change the size of an array once it is defined. It causes lots of difficulties while handling the group of objects. To overcome this drawback of arrays, Java Collection Framework is introduced from JDK 1.2.

Java Collections Hierarchy :

All the classes and interfaces related to Java collections are kept in java.util package. List, Set, Queue and Map are four top level interfaces of Java collection framework. All these interfaces (except Map) inherit from java.util.Collection interface which is the root interface in the Java collection framework.

List

Intro :

- List is a sequential collection of objects.
- Elements are positioned using zero-based index.
- Elements can be inserted or removed or retrieved from any arbitrary position using an integer index.

Popular Implementations :

- ArrayList, Vector And LinkedList

Internal Structure :

- **ArrayList** : Internally uses re-sizable array which grows or shrinks as we add or delete elements.
- **Vector** : Same as ArrayList but it is synchronized.
- **LinkedList** : Elements are stored as Nodes where each node consists of three parts – Reference To Previous Element, Value Of The Element and Reference To Next Element.

Null Elements :

- **ArrayList** : Yes
- **Vector** : Yes
- **LinkedList** : Yes

Duplicate Elements :

- **ArrayList** : Yes
- **Vector** : Yes
- **LinkedList** : Yes

Order Of Elements :

- **ArrayList** : Insertion Order
- **Vector** : Insertion Order
- **LinkedList** : Insertion Order

Synchronization :

- **ArrayList** : Not synchronized
- **Vector** : Synchronized
- **LinkedList** : Not synchronized

Performance :

- **ArrayList** : Insertion -> O(1) (If insertion causes restructuring of internal array, it will be O(n)), Removal -> O(1) (If removal causes restructuring of internal array, it will be O(n)), Retrieval -> O(1)
- **Vector** : Similar to ArrayList but little slower because of synchronization.
- **LinkedList** : Insertion -> O(1), Removal -> O(1), Retrieval -> O(n)

When to use?

- **ArrayList** : Use it when more search operations are needed then insertion and removal.
- **Vector** : Use it when you need synchronized list.
- **LinkedList** : Use it when insertion and removal are needed frequently.

Queue

Intro :

- Queue is a data structure where elements are added from one end called tail of the queue and elements are removed from another end called head of the queue.
- Queue is typically FIFO (First-In-First-Out) type of data structure.

Popular Implementations :

- PriorityQueue, ArrayDeque and LinkedList (implements List also)

Internal Structure :

- **PriorityQueue** : It internally uses re-sizable array to store the elements and a Comparator to place the elements in some specific order.
- **ArrayDeque** : It internally uses re-sizable array to store the elements.

Null Elements :

- **PriorityQueue** : Not allowed
- **ArrayDeque** : Not allowed

Duplicate Elements :

- **PriorityQueue** : Yes
- **ArrayDeque** : Yes

Order Of Elements :

- **PriorityQueue** : Elements are placed according to supplied Comparator or in natural order if no Comparator is supplied.
- **ArrayDeque** : Supports both LIFO and FIFO

Synchronization :

- **PriorityQueue** : Not synchronized
- **ArrayDeque** : Not synchronized

Performance :

- **PriorityQueue** : Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(1)
- **ArrayDeque** : Insertion -> O(1), Removal -> O(n), Retrieval -> O(1)

When to use?

- **PriorityQueue** : Use it when you want a queue of elements placed in some specific order.
- **ArrayDeque** : You can use it as a queue OR as a stack.

Set

Intro :

- Set is a linear collection of objects with no duplicates.
- Set interface does not have its own methods. All its methods are inherited from Collection interface. It just applies restriction on methods so that duplicate elements are always avoided.

Popular Implementations :

- HashSet, LinkedHashSet and TreeSet

Internal Structure :

- **HashSet** : Internally uses HashMap to store the elements.
- **LinkedHashSet** : Internally uses LinkedHashMap to store the elements.
- **TreeSet** : Internally uses TreeMap to store the elements.

Null Elements :

- **HashSet** : Maximum one null element
- **LinkedHashSet** : Maximum one null element.
- **TreeSet** : Doesn't allow even a single null element

Duplicate Elements :

- **HashSet** : Not allowed
- **LinkedHashSet** : Not allowed
- **TreeSet** : Not allowed

Order Of Elements :

- **HashSet** : No order
- **LinkedHashSet** : Insertion order
- **TreeSet** : Elements are placed according to supplied Comparator or in natural order if no Comparator is supplied.

Synchronization :

- **HashSet** : Not synchronized
- **LinkedHashSet** : Not synchronized
- **TreeSet** : Not synchronized

Performance :

- **HashSet** : Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **LinkedHashSet** : Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **TreeSet** : Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(log(n))

When to use?

- **HashSet** : Use it when you want only unique elements without any order.
- **LinkedHashSet** : Use it when you want only unique elements in insertion order.
- **TreeSet** : Use it when you want only unique elements in some specific order.

Map

Intro :

- Map stores the data in the form of key-value pairs where each key is associated with a value.
- Map interface is part of Java collection framework but it doesn't inherit Collection interface.

Popular Implementations :

- HashMap, LinkedHashMap And TreeMap

Internal Structure :

- **HashMap** : It internally uses an array of buckets where each bucket internally uses linked list to hold the elements.
- **LinkedHashMap** : Same as HashMap but it additionally uses a doubly linked list to maintain insertion order of elements.
- **TreeMap** : It internally uses Red-Black tree.

Null Elements :

- HashMap : Only one null key and can have multiple null values
- LinkedHashMap : Only one null key and can have multiple null values.
- TreeMap : Doesn't allow even a single null key but can have multiple null values.

Duplicate Elements :

- **HashMap** : Doesn't allow duplicate keys but can have duplicate values.
- **LinkedHashMap** : Doesn't allow duplicate keys but can have duplicate values.
- **TreeMap** : Doesn't allow duplicate keys but can have duplicate values.

Order Of Elements :

- **HashMap** : No Order
- **LinkedHashMap** : Insertion Order
- **TreeMap** : Elements are placed according to supplied Comparator or in natural order of keys if no Comparator is supplied.

Synchronization :

- **HashMap** : Not synchronized
- **LinkedHashMap** : Not Synchronized
- **TreeMap** : Not Synchronized

Performance :

- **HashMap** : Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **LinkedHashMap** : Insertion -> O(1), Removal -> O(1), Retrieval -> O(1)
- **TreeMap** : Insertion -> O(log(n)), Removal -> O(log(n)), Retrieval -> O(log(n))

When to use?

- **HashMap** : Use it if you want only key-value pairs without any order.
- **LinkedHashMap** : Use it if you want key-value pairs in insertion order.
- **TreeMap** : Use it when you want key-value pairs sorted in some specific order.

Java collections cheat sheet is an essential guide for developers working with the Java programming language. With the vast array of data structures available in the Java Collections Framework (JCF), understanding how to effectively utilize these collections can greatly enhance a programmer's ability to

manage and manipulate data efficiently. This cheat sheet covers the core components of the Java Collections Framework, including interfaces, implementations, and common methods, providing a comprehensive overview for both beginners and experienced developers.

Overview of Java Collections Framework

The Java Collections Framework is a unified architecture that provides a set of interfaces and classes to handle collections of objects. It allows developers to store, retrieve, and manipulate data in various ways, enabling more efficient programming practices.

Key Components

1. Interfaces: The JCF provides several key interfaces:
- Collection: The root interface from which all collections inherit. It defines basic operations like adding, removing, and checking for elements.
- List: An ordered collection that allows duplicates. Elements can be accessed by their integer index.
- Set: A collection that does not allow duplicates. It is used when uniqueness is a requirement.
- Map: A collection that stores key-value pairs. Keys are unique, while values can be duplicated.

2. Implementations: The JCF includes various implementations of these interfaces:
- ArrayList: A resizable array implementation of the List interface. It allows fast random access and is best for storing data that is frequently read.
- LinkedList: A doubly-linked list implementation of the List interface. It is more efficient for insertions and deletions than ArrayList.
- HashSet: An implementation of the Set interface that uses a hash table for storage. It allows fast access and does not maintain any order.
- TreeSet: A sorted implementation of the Set interface. It stores elements in a sorted manner based on their natural ordering or a specified comparator.
- HashMap: A hash table-based implementation of the Map interface. It allows fast key-value pair

access and is not synchronized.

- TreeMap: A sorted implementation of the Map interface that maintains its entries in ascending key order.

Common Operations

Understanding common operations available for collections is vital for effective programming. Below are some of the most frequently used methods across different collections.

List Operations

1. Adding Elements:

- `add(E e)`: Adds an element to the end of the list.
- `add(int index, E element)`: Inserts an element at the specified position.

2. Removing Elements:

- `remove(int index)`: Removes the element at the specified position.
- `remove(Object o)`: Removes the first occurrence of the specified element.

3. Accessing Elements:

- `get(int index)`: Retrieves the element at the specified position.
- `size()`: Returns the number of elements in the list.

Set Operations

1. Adding Elements:

- `add(E e)`: Adds the specified element to the set if it is not already present.

2. Removing Elements:

- `remove(Object o)`: Removes the specified element from the set.

3. Checking Membership:

- `contains(Object o)`: Returns true if the set contains the specified element.
- `size()`: Returns the number of elements in the set.

Map Operations

1. Adding Key-Value Pairs:

- `put(K key, V value)`: Associates the specified value with the specified key.
- `putAll(Map<? extends K, ? extends V> m)`: Copies all mappings from the specified map to this map.

2. Removing Key-Value Pairs:

- `remove(Object key)`: Removes the mapping for a specified key.

3. Accessing Values:

- `get(Object key)`: Returns the value to which the specified key is mapped.
- `keySet()`: Returns a set view of the keys contained in the map.

Iterating Over Collections

Java collections provide various ways to iterate through elements. The choice of iteration method can affect performance and readability.

Using For-Each Loop

The enhanced for-each loop is a simple way to iterate over collections:

```java
for (ElementType element : collection) {
// Process element
}
```

Using Iterator

The Iterator interface allows for safe removal of elements during iteration:

```java
Iterator iterator = collection.iterator();
while (iterator.hasNext()) {
ElementType element = iterator.next();
if (condition) {
iterator.remove(); // Safely remove element
}
}
```

Using Streams

Java 8 introduced the Stream API, which provides a functional approach to processing collections:

```java
collection.stream()
.filter(element -> condition)
```

.forEach(element -> process(element));
```

## Performance Considerations

When choosing which collection to use, performance is a crucial factor. Different collections have different time complexities for operations.

### ArrayList vs. LinkedList

- ArrayList:
- Access: O(1)
- Insertion/Deletion: O(n) (due to array resizing)

- LinkedList:
- Access: O(n)
- Insertion/Deletion: O(1) (when the position is known)

### HashSet vs. TreeSet

- HashSet:
- Access: O(1) on average
- Order: Unordered

- TreeSet:
- Access: O(log n)
- Order: Sorted
```

HashMap vs. TreeMap

- HashMap:
- Access: O(1) on average
- Order: Unordered

- TreeMap:
- Access: O(log n)
- Order: Sorted by keys

Common Use Cases

Choosing the right collection based on use cases can greatly improve performance and code clarity.

1. ArrayList: Use when you need fast access and don't need to frequently insert or delete elements.
2. LinkedList: Ideal for scenarios where you need frequent insertions and deletions.
3. HashSet: Best for storing unique elements where order does not matter.
4. TreeSet: Choose when you need a sorted set of elements.
5. HashMap: Useful for fast lookups and storing key-value pairs where order is not important.
6. TreeMap: When you need to maintain a sorted order of keys in a map.

Conclusion

The Java collections cheat sheet serves as a foundational resource for understanding the Java Collections Framework. By grasping the core interfaces, their implementations, common operations, and performance implications, developers can make informed decisions when handling collections. Whether you're building simple applications or complex systems, mastering Java collections is essential for effective data manipulation and management in Java. With ongoing practice and

application of these concepts, developers will find themselves more adept at leveraging the full power of the Java Collections Framework.

Frequently Asked Questions

What are Java Collections?

Java Collections are a framework that provides an architecture for storing and manipulating a group of objects. They include classes and interfaces for data structures like lists, sets, and maps.

What is the difference between List, Set, and Map in Java Collections?

A List is an ordered collection that allows duplicate elements, a Set is a collection that does not allow duplicates, and a Map is a collection of key-value pairs where each key is unique.

What is the purpose of the Collections utility class in Java?

The Collections utility class provides static methods for operations on collections, such as searching, sorting, and synchronizing collections.

How do you convert an array to a List in Java?

You can convert an array to a List using the Arrays.asList() method. For example: List<String> list = Arrays.asList(array);

What are the main implementations of the List interface in Java?

The main implementations of the List interface are ArrayList, LinkedList, and Vector. Each has different performance characteristics and use cases.

How can you iterate over a Map in Java?

You can iterate over a Map using the keySet(), entrySet(), or values() methods. For example, using entrySet(): for (Map.Entry<KeyType, ValueType> entry : map.entrySet()) { ... }

Java Collections Cheat Sheet

自学 *Java* 怎么入门？ - 知乎
在大学自学 Java，看视频教程和代码能看懂，自己写就写不出来，怎么样能够入门？

预测一下**2025**年**Java**就业趋势？ **-** 知乎
Jan 6, 2025 · Java曾经是IT行业最大的就业岗位，但是现在这个行业马上就要没了，一本的软件工程专业搞java得就业率还不到30%，未来几年java都不会起来了。

Java开发者社区**-CSDN**社区云
Dec 30, 2024 · 深知自学Java很难，所以最近整理了这份Java学习路线，适配2023年新版，是我自学Java时跟着学的路线，绝对是Java入门最佳选择 当然，你可能不需要学习全部的视频，可 …

Java LTS版本有哪些？ **-** 知乎
Java LTS版本（长期支持版本）对于企业和开发者来说至关重要，能提供稳定的开发和生产环境，并在较长时间内获得官方支持，包括安全更新、Bug修复和性能提升，目前主要的Java LTS版本 …

Java社区-CSDN社区云
CSDNJava社区,Java论坛,为中国软件开发者打造学习和成长的家园

*Java*真的是要没落了吗？*2024*年还有希望吗？ - 知乎
Java真的是要没落了吗？ 2024年还有希望吗？作为SpringCloudAlibaba微服务架构实战派上下册和RocketMQ消息中间件实战派上下册的作者胡弦，最近很多从事Java的技术小伙伴都跑… 显 …

Java后端技术壁垒有哪些？ **-** 知乎
1 单机版的Java后端，比如基于spring boot的增删改查，中专生经过培训，半年能写很熟，外加能解决问题，这块没有技术壁垒。 2 顺带第1点说出去，JavaEE（就集合异常处理等）部分 …

A Java Exception has occurred.怎么解决啊**…-CSDN**社区
Feb 7, 2010 · 解决打包后双击提示"a java exception has occurred"的问题了。方法是删掉1.7版本的jdk，换上1.6版本的jdk（虽然我不确定此问题跟jdk有关）。换jdk版本后eclipse会出现错误 …

求助!!! JDK双击没反应!-CSDN社区
Jun 2, 2014 · 以下内容是CSDN社区关于求助!!! JDK双击没反应!相关内容，如果想了解更多关于Java SE社区其他内容，请访问CSDN社区。

Spring Boot使用**Redis**的**Lettuce**连接池，隔一段时间再连接就报连 **…**
Apr 13, 2019 · 以下内容是CSDN社区关于Spring Boot使用Redis的Lettuce连接池，隔一段时间再连接就报连接超时相关内容，如果想了解更多关于Java社区其他内容，请访问CSDN社区。

自学 **Java** 怎么入门？ **-** 知乎
在大学自学 Java，看视频教程和代码能看懂，自己写就写不出来，怎么样能够入门？

预测一下**2025**年**Java**就业趋势？ **-** 知乎
Jan 6, 2025 · Java曾经是IT行业最大的就业岗位，但是现在这个行业马上就要没了，一本的软件工程专业搞java得就业率还不到30%，未来几年java都不会起来了。

Java开发者社区-CSDN社区云
Dec 30, 2024 · 深知自学Java很难，所以最近整理了这份Java学习路线，适配2023年新版，是我自学Java时跟着学的路线，绝对是Java入门最佳选择 当然，你可能不需要学习全部的视频，可 ...

Java LTS版本有哪些？ - 知乎
Java LTS版本（长期支持版本)对于企业和开发者来说至关重要，能提供稳定的开发和生产环境，并在较长时间内获得官方支持，包括安全更新、Bug修复和性能提升，目前主要的Java LTS版本 ...

Java社区-CSDN社区云
CSDNJava社区,Java论坛,为中国软件开发者打造学习和成长的家园

Java真的是要没落了吗？2024年还有希望吗？ - 知乎
Java真的是要没落了吗？ 2024年还有希望吗？ 作为SpringCloudAlibaba微服务架构实战派上下册和RocketMQ消息中间件实战派上下册的作者胡弦，最近很多从事Java的技术小伙伴都跑... 显 ...

*Java*后端技术壁垒有哪些？ - 知乎
1 单机版的Java后端，比如基于spring boot的增删改查，中专生经过培训，半年能写很熟，外加能解决问题，这块没有技术壁垒。 2 顺带第1点说出去，JavaEE（就集合异常处理等）部分 ...

A Java Exception has occurred.怎么解决啊...-CSDN社区
Feb 7, 2010 · 解决打包后双击提示"a java exception has occurred"的问题了。方法是删掉1.7版本的jdk，换上1.6版本的jdk（虽然我不确定此问题跟jdk有关）。换jdk版本后eclipse会出现错误 ...

求助!!! JDK双击没反应!-CSDN社区
Jun 2, 2014 · 以下内容是CSDN社区关于求助!!! JDK双击没反应!相关内容，如果想了解更多关于Java SE社区其他内容，请访问CSDN社区。

Spring Boot使用Redis的Lettuce连接池，隔一段时间再连接就报连 ...
Apr 13, 2019 · 以下内容是CSDN社区关于Spring Boot使用Redis的Lettuce连接池，隔一段时间再连接就报连接超时相关内容，如果想了解更多关于Java社区其他内容，请访问CSDN社区。

Unlock the power of Java with our comprehensive Java collections cheat sheet! Master essential data structures and improve your coding skills. Learn more!

Back to Home