


Java Data Structures Cheat Sheet

Java 8 Best Practices Cheat Sheet



Default methods

Evolve interfaces & create traits

```
//Default methods in interfaces
//FunctionalInterface
interface Utilities {
    default Consumer<Runnable> m() {
        return (r) -> r.run();
    }
}
// default methods, still functional
Object function(Object o) {}

class A implements Utilities { // implement
    public Object function(Object o) {
        return new Object();
    }
}
// call a default method
Consumer<Runnable> m = new A().m();
}
```

Lambdas

Syntax:
(parameters)-> expression
(parameters)-> { statements; }

```
// takes a long, returns a String
Function<Long, String> f = l -> l.toString();
// takes nothing gives you Thread
Supplier<Thread> s = Thread::currentThread;
// takes a string as the parameter
Consumer<String> c = System.out::println;

// use them with streams
new ArrayList<String>().stream()
    // peek: debug streams without changes
    peek(e -> System.out.println(e))
    // map: convert every element into something
    map(e -> e.hashCode())
    // filter: pass some elements through
    filter(e -> ((e.hashCode() % 2) == 0))
    // collect all values from the stream
    collect(Collectors.toCollection(TreeSet::new));
```

java.util.Optional

A container for possible null values

```
// Create an optional
Optional<String> optional =
    Optional.ofNullable(a);
// process the optional
optional.map(s -> "Rebellabs:" + s);
// map a function that returns Optional
optional.flatMap(s -> Optional.ofNullable(s));

// run if the value is there
optional.ifPresent(System.out::println);
// get the value or throw an exception
optional.get();

// return the value or the given value
optional.orElse("Hello world!");
// return empty Optional if not satisfied
optional.filter(s -> s.startsWith("Rebellabs"));
```

Rules of Thumb

Traits: 1 default method per interface
Don't enhance functional interfaces
Only conservative implementations

Expressions over statements
Refactor to use method references
Chain lambdas rather than growing them

Fields - use plain objects
Method parameters - use plain objects
Return values - use Optional
Use `orElse()` instead of `get()`

Java data structures cheat sheet serves as a handy reference for developers, programmers, and students who want to understand the various data structures available in Java and how to use them effectively. Java provides a rich set of built-in data structures that can help in organizing and managing data efficiently. This article will cover the most common data structures found in Java, their characteristics, use cases, and code examples.

Introduction to Data Structures in Java

Data structures are fundamental to computer science and programming, as they provide a way of organizing, managing, and storing data. Choosing the right data structure for a specific problem can significantly affect the performance and efficiency of an application. In Java, data structures can be broadly categorized into two types:

- Primitive Data Structures
- Non-Primitive Data Structures

Primitive data structures include basic types like integers, floats, and characters, while non-primitive data structures include arrays, strings, classes, and collections.

Java Collections Framework

The Java Collections Framework (JCF) provides a set of classes and interfaces for working with groups of objects. It includes data structures for storing and manipulating data. The key interfaces in the JCF are:

- **Collection**
- **List**
- **Set**
- **Map**
- **Queue**

Each of these interfaces has different implementations, allowing developers to choose the most suitable one for their needs.

1. List

The List interface allows for ordered collections that can contain duplicate elements. The main implementations of the List interface are:

- **ArrayList**
- **LinkedList**
- **Vector**

ArrayList:

- Characteristics: Resizable array implementation; allows random access.
- Use Cases: When frequent access and iteration through elements are required.
- Example:

```
```java
ArrayList list = new ArrayList<>();
list.add("Java");
list.add("Python");
list.add("C++");
```
```

LinkedList:

- Characteristics: Doubly linked list implementation; allows for efficient insertion and deletion.
- Use Cases: When the application requires frequent modifications.
- Example:

```
```java
LinkedList linkedList = new LinkedList<>();
linkedList.add("Java");
linkedList.addFirst("Python");
linkedList.addLast("C++");
```
```

Vector:

- Characteristics: Synchronized and dynamic array.
- Use Cases: When thread safety is a concern.
- Example:

```
```java
Vector vector = new Vector<>();
vector.add("Java");
vector.add("Python");
```
```

2. Set

The Set interface represents a collection that cannot contain duplicate elements. The main implementations of the Set interface are:

- **HashSet**
- **LinkedHashSet**
- **TreeSet**

HashSet:

- Characteristics: Unordered collection; allows null elements.
- Use Cases: When uniqueness is required without maintaining order.
- Example:

```
```java
HashSet hashSet = new HashSet<>();
hashSet.add("Java");
hashSet.add("Python");
```
```

LinkedHashSet:

- Characteristics: Maintains insertion order; allows null elements.
- Use Cases: When order matters alongside uniqueness.
- Example:

```
```java
LinkedHashSet linkedHashSet = new LinkedHashSet<>();
linkedHashSet.add("Java");
linkedHashSet.add("Python");
```
```

TreeSet:

- Characteristics: Navigable set; elements are stored in a sorted manner.
- Use Cases: When sorted order is required.
- Example:

```
```java
TreeSet treeSet = new TreeSet<>();
treeSet.add("Java");
treeSet.add("Python");
```
```

3. Map

The Map interface represents a collection of key-value pairs, where each key is unique. The main implementations of the Map interface are:

- **HashMap**
- **LinkedHashMap**
- **TreeMap**

HashMap:

- Characteristics: Unsynchronized; allows null values and one null key.
- Use Cases: When fast retrieval of data based on keys is needed.
- Example:

```
```java
HashMap hashMap = new HashMap<>();
hashMap.put("Java", 1);
hashMap.put("Python", 2);
```
```

LinkedHashMap:

- Characteristics: Maintains insertion order; allows null keys and values.
- Use Cases: When order matters alongside key-value association.

- Example:

```
```java
LinkedHashMap linkedHashMap = new LinkedHashMap<>();
linkedHashMap.put("Java", 1);
linkedHashMap.put("Python", 2);
```
```

TreeMap:

- Characteristics: Sorted map; keys are stored in a natural order.

- Use Cases: When a sorted key-value association is necessary.

- Example:

```
```java
TreeMap treeMap = new TreeMap<>();
treeMap.put("Java", 1);
treeMap.put("Python", 2);
```
```

4. Queue

The Queue interface represents a collection designed for holding elements prior to processing. The main implementations of the Queue interface are:

- **LinkedList** (also implements Queue)
- **PriorityQueue**
- **ArrayDeque**

LinkedList:

- Characteristics: Can act as a queue; allows elements to be added and removed from both ends.

- Use Cases: When a queue with flexible capacity and operations is needed.

- Example:

```
```java
Queue queue = new LinkedList<>();
queue.add("Java");
queue.add("Python");
```
```

PriorityQueue:

- Characteristics: Elements are ordered according to their natural ordering or a comparator; does not allow null.

- Use Cases: When processing elements in priority order is required.

- Example:

```
```java
PriorityQueue priorityQueue = new PriorityQueue<>();
priorityQueue.add(3);
priorityQueue.add(1);
```
```

ArrayDeque:

- Characteristics: Resizable-array implementation of the Deque interface; allows elements to be added and removed from both ends.

- Use Cases: When a double-ended queue is needed.

- Example:

```
```java
ArrayDeque arrayDeque = new ArrayDeque<>();
arrayDeque.add("Java");
arrayDeque.addFirst("Python");
```
```

Conclusion

Understanding the various Java data structures and their characteristics is essential for any developer looking to write efficient and effective code. The Java Collections Framework provides a comprehensive set of data structures that can be leveraged for different scenarios. By choosing the appropriate data structure, developers can optimize performance and enhance the functionality of their applications. Keep this cheat sheet handy as you work on your Java projects, and you will find it easier to select the right tools for your programming tasks.

Frequently Asked Questions

What is a Java data structures cheat sheet?

A Java data structures cheat sheet is a quick reference guide that summarizes the key data structures available in Java, including their properties, operations, and use cases.

What are the most commonly used data structures in Java?

The most commonly used data structures in Java include Arrays, LinkedLists, Stacks, Queues, HashMaps, and Trees.

How can I effectively use a cheat sheet for learning Java

data structures?

To effectively use a cheat sheet, refer to it while coding to understand the syntax and methods of each data structure, practice implementing them, and review the cheat sheet regularly to reinforce your knowledge.

Are there any online resources for Java data structures cheat sheets?

Yes, there are numerous online resources, including websites like GitHub, educational platforms, and coding blogs that provide downloadable or interactive Java data structures cheat sheets.

What should be included in a comprehensive Java data structures cheat sheet?

A comprehensive cheat sheet should include the definitions, time and space complexities, common methods, examples of use, and comparisons of different data structures.

Can I use a data structures cheat sheet for interview preparation?

Absolutely! A data structures cheat sheet is an excellent tool for interview preparation, as it helps you quickly recall essential concepts and algorithms often discussed in technical interviews.

Find other PDF article:

<https://soc.up.edu.ph/60-flick/pdf?docid=KRU91-5014&title=the-measure-of-a-man-by-sidney-poitier.pdf>

Java Data Structures Cheat Sheet

Java 数据结构和算法 - 文档

Java 数据结构和算法 - 文档

2025 Java 数据结构和算法 - 文档

Jan 6, 2025 · Java 数据结构和算法 - 文档

Java 数据结构和算法 - CSDN 文档

Dec 30, 2024 · Java 数据结构和算法 - 文档

Java LTS 数据结构和算法 - 文档

Java LTS 数据结构和算法 - 文档

Java- CSDN
CSDNJava,Java,...

Java2024 -
Java 2024 SpringCloudAlibabaRocketMQ
Java... ..

Java -
1 Java spring boot 2 1JavaEE
...

A Java Exception has occurred.- CSDN
Feb 7, 2010 · "a java exception has occurred" 1.7jdk1.6jdk
jdk jdk eclipse ...

!!! JDK!- CSDN
Jun 2, 2014 · CSDN!!! JDK!Java SE CSDN

Spring BootRedisLettuce ...
Apr 13, 2019 · CSDNSpring BootRedisLettuce
Java CSDN

Java -
Java

2025Java -
Jan 6, 2025 · JavaITjava30%java

Java- CSDN
Dec 30, 2024 · JavaJava2023JavaJava

Java LTS -
Java LTS () Bug
Java LTS ...

Java- CSDN
CSDNJava,Java,...

Java2024 -
Java 2024 SpringCloudAlibabaRocketMQ
Java... ..

Java -
1 Java spring boot 2 1JavaEE
...

A Java Exception has occurred.- CSDN
Feb 7, 2010 · "a java exception has occurred" 1.7jdk1.6jdk
jdk jdk eclipse ...

Jun 2, 2014 · CSDN!!! JDK!Java SE CSDN

Apr 13, 2019 · [CSDN](#) [Spring Boot](#) [Redis](#) [Lettuce](#)

[Back to Home](#)