**Java and Java 8 Notes**

**1) OOPs principles- Inheritance Encapsulation Polymorphism Abstraction**

**2) Method overloading vs method overriding (Explain rules).**

Overloading Overriding

- can be done in same class

-Inheritance Is not required

- same method name in same calss

--number of args diff

-- type of args diff

-- sequence of args diff

**3) Why string objects are immutable?(why strigs have been made immutable in java)**

Ans: Once we create a String object we can't perform any change in existing object. If we are trying to perform any change with those changes a new object will be created. This non-changeable behaviour is nothing but immutability of String.

In general, strings are used to represent the vital details like database connection URLs, username passwords etc. The immutability of strings helps to keep such details unchanged.

Similarly, String is used as an argument while loading the class. At that time change in the string may result in loading the wrong class. If immutable automatically variables (strings) are thread safe.

Immutable Strings increase security. As they can't be modified once they have created, so we can use them to store sensitive data like username, pw etc. Immutable strings are thread safe so we can use them in a multi-threaded code without synchronisation.

4) **StringBuffer vs StringBuilder**

🡪String Buffer String Builder

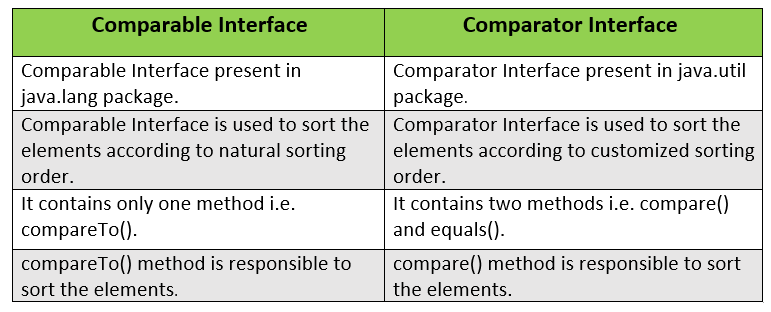
* Thread safe -Non thread safe
* (two threads can’t call the

methods of stringBuffer simultaneously)

* Synchronised - non- synchronised
* Slower and less efficient than - faster and more efficient

string builder

**5) Comparable vs Comparator. Advantages of Comparator**



* A class must implement the comparable interface if sorting required on collection of objects, but, for comparator we don’t need to implement it.
* By using comparator, objects can be sorted based on more than one fields of a class. Whereas Comparable allows you to sort items in a collections based on only one field.

**6) How put() method of HashMap works?**

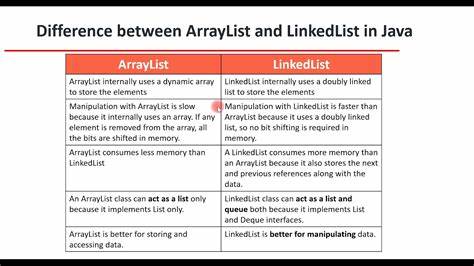
Ans: put() method of HashMap is used to insert a mapping into a map. This means we can insert a specific key and the value it is mapping to into a particular map. If an existing key is passed then the previous value gets replaced by the new value. If a new pair is passed, then the pair gets inserted as a whole.

**7) Importance of equals() and hashcode() method. What is a contract between equals() and hashcode() method?**

🡪 The equals () and hashcode () are the two important methods provided by the Object class for**comparing objects**. Since the Object class is the parent class for all Java objects, hence all objects inherit the default implementation of these two methods.

**8) Explain TreeSet.**

**9) Difference between ArrayList and LinkedList. Use cases of ArrayList and LinkedList.**

Ans: 

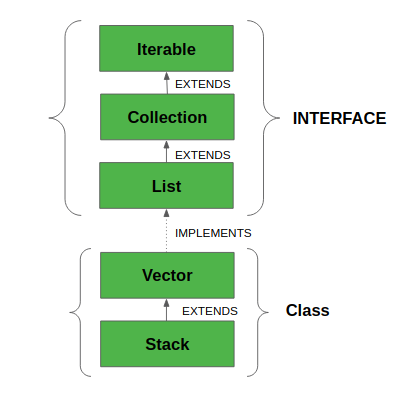
**Use cases: ArrayList is better for storing and accessing data whereas LinkedList is better for Manipulating data.**

Manipulating LinkedList takes less time compared to ArrayList because, in a doubly-linked list, there is no concept of shifting the memory bits. The list is traversed and the reference link is changed. This class implements a List interface. Therefore, this acts as a list. This class implements both the List interface and the Deque interface.

**10) Why Vector and Stack class are not used?**

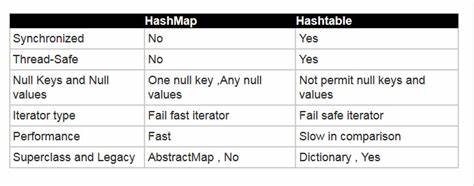
Ans: The Vector class implements a growable array of objects. Vectors fall in legacy classes, but now it is fully compatible with collections. It is found in[java.util package](https://www.geeksforgeeks.org/java-util-package-java/) and implement the [List](https://www.geeksforgeeks.org/list-interface-java-examples/) interface, so we can use all the methods of the List interface as shown below as follows:

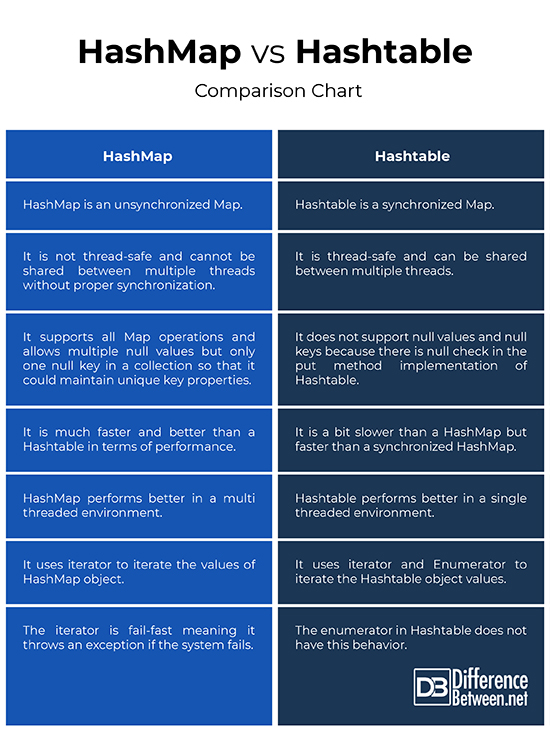
Also stack’s working is similar to deque.



* Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.
* They are very similar to [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/), but Vector is synchronized and has some legacy methods that the collection framework does not contain.
* It also maintains an insertion order like an ArrayList. Still, it is rarely used in a non-thread environment as it is **synchronized**, and due to this, it gives a poor performance in adding, searching, deleting, and updating its elements.
* The Iterators returned by the Vector class are fail-fast. In the case of concurrent modification, it fails and throws the **ConcurrentModificationException.**

**11) Difference between HashMap and Hashtable.**





**12) Why there is a category of Checked Exceptions? Why some exceptions are checked and others are unchecked? (Explain why)**

Ans:

12) Explain Java 8 new features.

**13) Advantages of Lambda Expression. Before Java 8 what you were using??**

**Ans: Fewer Lines of Code** − One of the most benefits of a lambda expression is to **reduce the amount of code**. We know that lambda expressions can be used only with a **functional interface**. For instance, **Runnable**is a functional interface, so we can easily apply lambda expressions.

**Sequential and Parallel execution support by passing behavior as an argument in methods** − By using **Stream API** in Java 8, the functions are passed to collection methods. Now it is the responsibility of collection for processing the elements either in a sequential or parallel manner.

**Higher Efficiency** − By using **Stream API** and **lambda expressions**, we can achieve higher efficiency (**parallel execution**) in case of bulk operations on collections. Also, lambda expression helps in achieving the internal iteration of collections rather than external iteration.

// Before Java 8:

new Thread(new Runnable() {

@Override

public void run() {

System.out.println("Before Java8, too much code for too little to do");

}

}).start();

//Java 8 way:

new Thread( () -> System.out.println("In Java8, Lambda expression rocks !!") ).start();

**14) What is a functional interface? In which package functional interfaces are introduced in Java 8? Explain categories of functional interfaces.**

Ans: Interface contains only one abstract method. They may have default and static methods.

Java 8 has also introduced new package => java.util.function

public interface Calculator {

public int calculate(int x, int y);

}

##1st Way

public class CalculatorImpl implements Calculator {

@Override

public int calculate(int x, int y) {

return x + y;

}

}

##2nd Way => Anonymous Inner class.

Calculator c = new Calculator() {

@Override

public int calculate(int x, int y) {

return x + y;

}

}

---

public interface Consumer<T> {

public void accept(T t);

}

public interface Predicate<T> {

public boolean test(T t);

}

public interface Supplier<T> {

public T get();

}

public interface Function<T, R> {

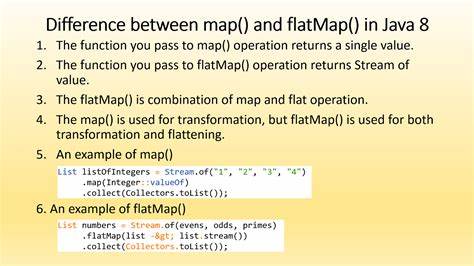
public R apply(T t);

}

**15) Explain use of Stream API.**

Stream API is used to**process collections of objects**. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result. A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.

16) Difference between map() and flatMap() method.



17) Explain intermediate and terminal operations in stream api.

18) Prepare most commonly used functional interfaces in Stream API.

(hint: Interface and its abstract method. For eg. Predicate<T> interface has "public boolean test(T t) method.)

19) Explain default and static methods of interface.

20) Difference between throw and throws. New features added in Java 7 related to exceptions (Hint: try with resources and multicatch block)

21) How to create custom exception?

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

|  |  |
| --- | --- |
| try | The "try" keyword is used to specify a block where we should place an exception code.  It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

**22) Explain exception chaining. (Hint - initCause() method)**

1. Lambda Expression

2. Stream API

3. Default and static methods in Interfaces

4. Functional interfaces- Having only one abstract method

5. Optional class-

**6. New Date and Time API-**

The new Date and Time APIs are**thread-safe, immutable, cacheable**, and represent a point in time measured to the **nanosecond** and have the option for backward compatibility. It borrows the ideas from Joda-Time and allows the programmers to capitalize on the features which were not available in java.util.Date and Calendar.

We can perform lot of **utility operations** on new date and time api.

Drawbacks of existing Date/Time API's

1. **Thread safety:** The existing classes such as Date and Calendar does not provide thread safety. Hence it leads to hard-to-debug concurrency issues that are needed to be taken care by developers. The new Date and Time APIs of Java 8 provide thread safety and are immutable, hence avoiding the concurrency issue from developers.
2. **Bad API designing:** The classic Date and Calendar APIs does not provide methods to perform basic day-to-day functionalities. The Date and Time classes introduced in Java 8 are ISO-centric and provides number of different methods for performing operations regarding date, time, duration and periods.
3. **Difficult time zone handling:** To handle the time-zone using classic Date and Calendar classes is difficult because the developers were supposed to write the logic for it. With the new APIs, the time-zone handling can be easily done with Local and ZonedDate/Time APIs.

Q. Why we don't use Stack class?

Inconsistency: Stack extends the Vector class, which allows you to access element by index. This is inconsistent with what a Stack should actually do, which is why the Deque interface is preferred (it does not allow such operations)--its allowed operations are consistent with what a FIFO or LIFO data structure should allow.

**Q. How put() method of map works?**

This method has to pass two arguments, key and value where key is the left argument and value is the corresponding value of the key in the map.

**Return Value:** If an existing key is passed then the previous value gets returned. If a new pair is passed, then NULL is returned.

String returned\_value = (String)hash\_map.put(20, "All");

7. Functional Programming & Why Functional Programming?

8. Method reference, Constructor reference

Q. When we deal with Set or Map collections, it is recommended to override equals() and hashcode() method, why so?