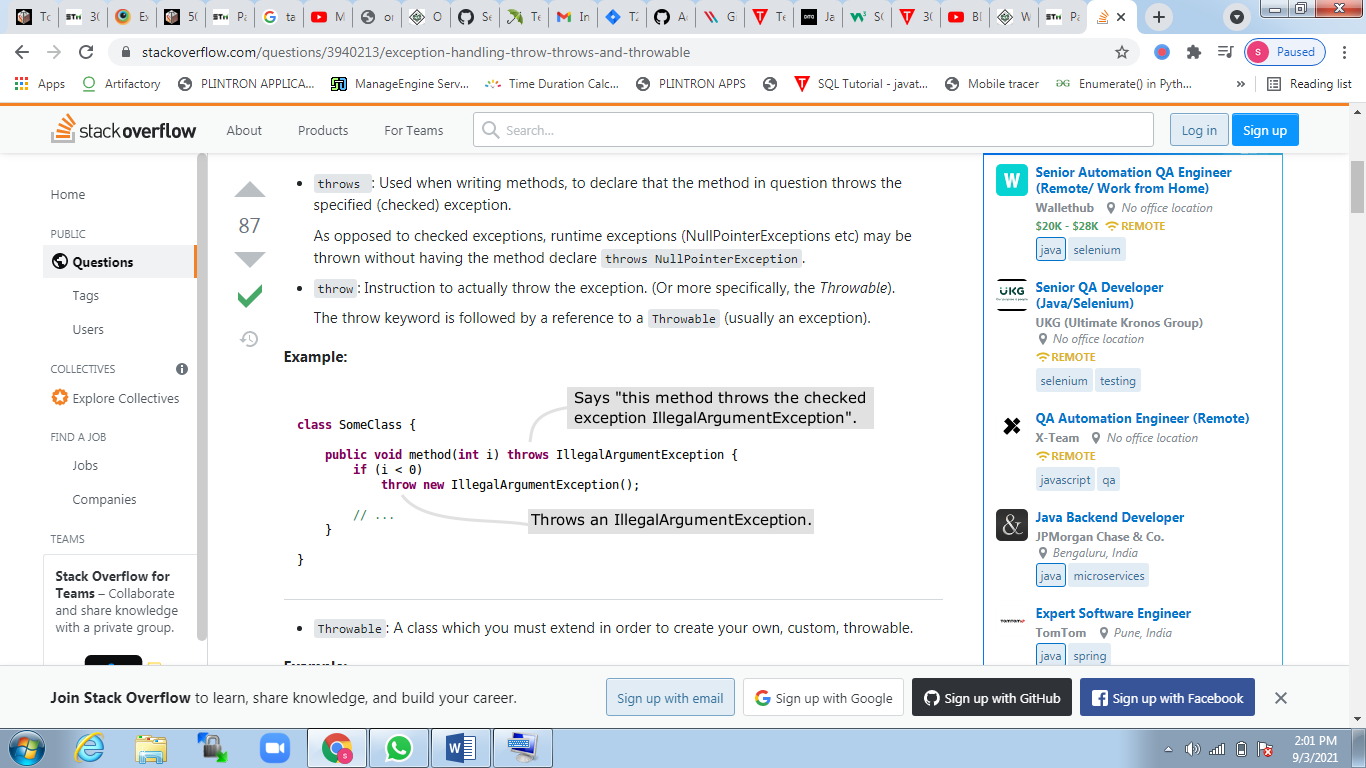
Questions

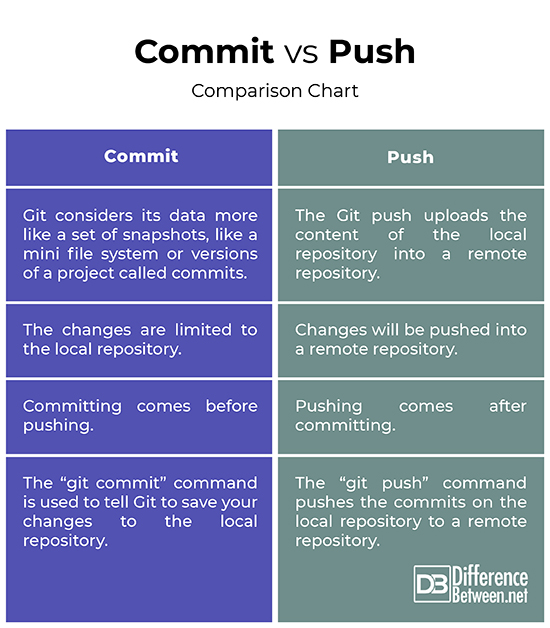


Final,finally,finalize

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. no.** | **Key** | **final** | **finally** | **finalize** |
| 1. | Definition | final is the keyword and access modifier which is used to apply restrictions on a class, method or variable. | finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not. | finalize is the method in Java which is used to perform clean up processing just before object is garbage collected. |
| 2. | Applicable to | Final keyword is used with the classes, methods and variables. | Finally block is always related to the try and catch block in exception handling. | finalize() method is used with the objects. |
| 3. | Functionality | (1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited. | (1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block | finalize method performs the cleaning activities with respect to the object before its destruction. |
| 4. | Execution | Final method is executed only when we call it. | Finally block is executed as soon as the try-catch block is executed.  It's execution is not dependant on the exception. | finalize method is executed just before the object is destroyed. |

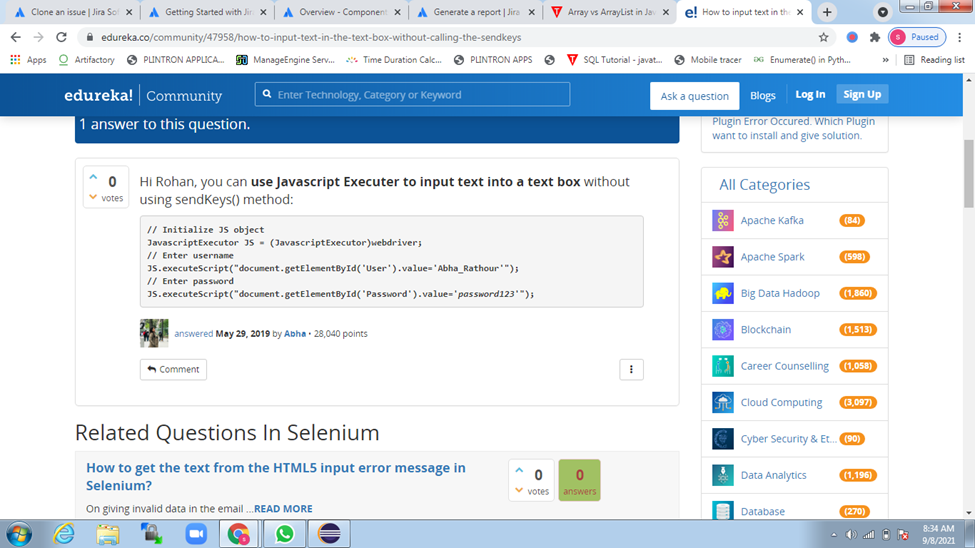
Final finally finalize

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. no.** | **Basis of Differences** | **throw** | **throws** |
| 1. | Definition | Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code. | Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code. |
| 2. | Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only. | Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only. |  |
| 3. | Syntax | The throw keyword is followed by an instance of Exception to be thrown. | The throws keyword is followed by class names of Exceptions to be thrown. |
| 4. | Declaration | throw is used within the method. | throws is used with the method signature. |
| 5. | Internal implementation | We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions. | We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException. |
|  |  |  |  |

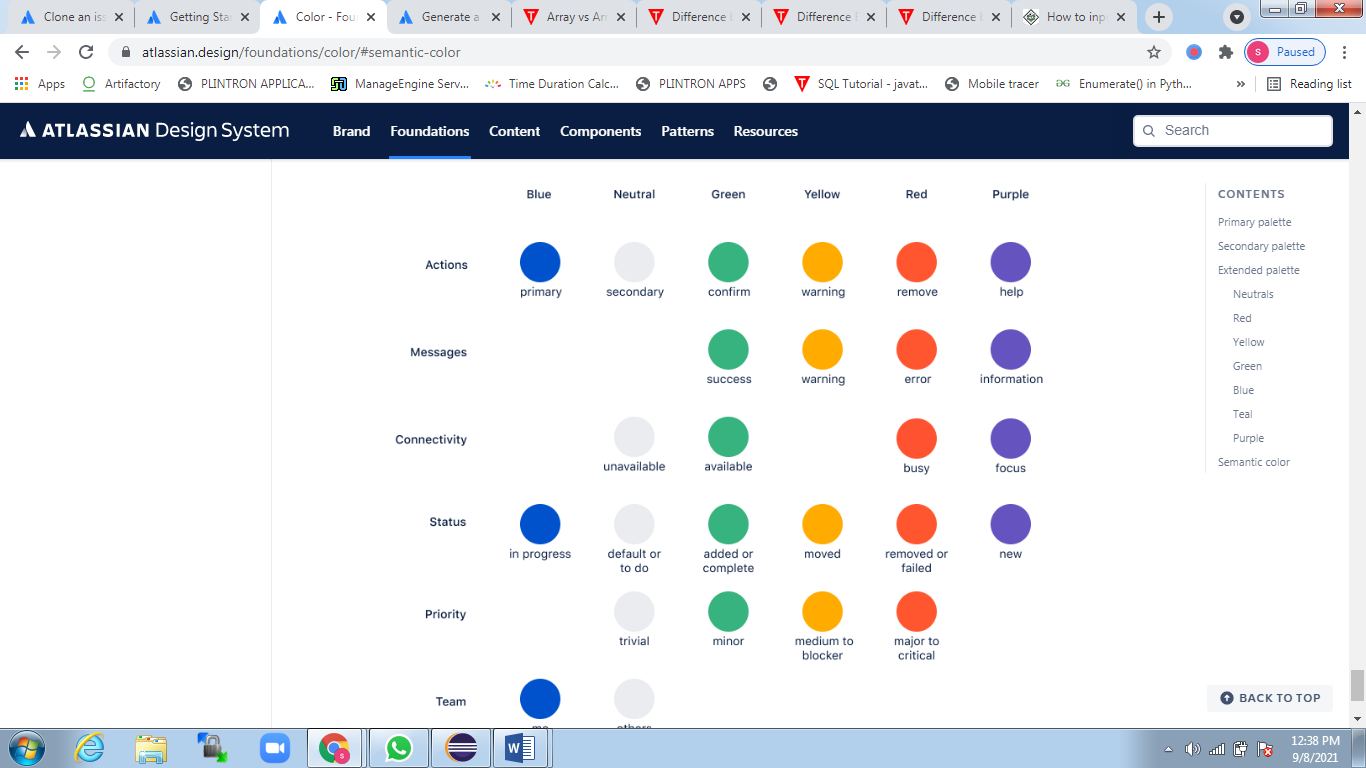


|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

|  |  |  |
| --- | --- | --- |
| **S.No** | **List** | **Set** |
| 1. | The list implementation allows us to add the same or duplicate elements. | The set implementation doesn't allow us to add the same or duplicate elements. |
| 2. | The insertion order is maintained by the List. | It doesn't maintain the insertion order of elements. |
| 3. | List allows us to add any number of null values. | Set allows us to add at least one null value in it. |
| 4. | The List implementation classes are LinkedList and ArrayList. | The Set implementation classes are TreeSet, HashSet and LinkedHashSet. |
| 5. | We can get the element of a specified index from the list using the get() method. | We cannot find the element from the Set based on the index because it doesn't provide any get method(). |
| 6. | It is used when we want to frequently access the elements by using the index. | It is used when we want to design a collection of distinct elements. |
| 7. | The method of List interface listiterator() is used to iterate the List elements. | The iterator is used when we need to iterate the Set elements. |



|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. |
| 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. |
| 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. |
| 4) HashMap is **fast**. | Hashtable is **slow**. |
| 5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap); | Hashtable is internally synchronized and can't be unsynchronized. |
| 6) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. |
| 7) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. |
| 8) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. |



**Why is string immutable in Java?**

The String is immutable in Java because **of the security, synchronization and concurrency, caching, and class loading**. The reason of making string final is to destroy the immutability and to not allow others to extend it. The String objects are cached in the String pool, and it makes the String immutable.

Wrapper classes in Java

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

* **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
* **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
* **Synchronization:** Java synchronization works with objects in Multithreading.
* **java.util package:** The java.util package provides the utility classes to deal with objects.
* **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

# Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve*[*abstraction*](https://www.javatpoint.com/abstract-class-in-java)

. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java)

.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

**Whats is a thread?**

Definition: A thread is **a single sequential flow of control within a program**.

Multithreading is a Java feature that **allows concurrent execution of two or more parts** of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process. Threads can be created by using two mechanisms

# Synchronization in Java

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

### **Why use Synchronization?**

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

**Which is collection in Java?**

The collection in java is **the root interface of the collection framework** and provide several classes and interfaces to represent a group of individual objects as a single unit. List, Set, and Queue are the main child interfaces of the collection interface

Diff btvn array & collection

| **Sr. No.** | **Key** | **Arrays** | **Collection** |
| --- | --- | --- | --- |
| 1 | Size | Arrays are fixed in size i.e once the array with the specific size is declared then we can't alter its size afterward. | The collection is dynamic in size i.e based on requirement size could be get altered even after its declaration. |
| 2 | Memory Consumption | Arrays due to fast execution consumes more memory and has better performance. | Collections, on the other hand, consume less memory but also have low performance as compared to Arrays. |
| 3 | Data type | Arrays can hold the only the same type of data in its collection i.e only homogeneous data types elements are allowed in case of arrays. | Collection, on the other hand, can hold both homogeneous and heterogeneous elements. |
| 4 | Primitives storage | Arrays can hold both object and primitive type data. | On the other hand, collection can hold only object types but not the primitive type of data. |
| 5 | Performance | Arrays due to its storage and internal implementation better in performance. | Collection on the other hand with respect to performance is not recommended to use. |

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Which is faster CSS or XPath?

**CSS is faster than XPath**. ... It is very useful when we want to test our application on multiple browsers because CSS engine are consistent in all browsers. CSS is best for IE as XPath does not work in IE always.

What is meant by constructor?

A constructor is **a special method of a class or structure in object-oriented programming that initializes a newly created object of that type**. Whenever an object is created, the constructor is called automatically

WebElement eleSearch = driver.findElement(By.xpath("//\*[@class='navsearchbar']//div[2]//div"));

String rgbFormat = eleSearch.getCssValue("background-color");

System.out.println(rgbFormat); //In RGB Format the value will be print => rgba(254, 189, 105, 1)

String hexcolor = Color.fromString(rgbFormat).asHex(); //converted Into HexFormat

System.out.println(hexcolor);// Output of Hex code will be => #febd69

The above code will return value in **RGB** format such as **“rgba(36, 93, 193, 1)”**.

Now we will convert it into **hexadecimal code** using **Java** as shown below:

***String[] hexValue = color.replace(“rgba(“, “”).replace(“)”, “”).split(“,”);***

***hexValue[0] = hexValue[0].trim();***

***int hexValue1 = Integer.parseInt(hexValue[0]);***

***hexValue[1] = hexValue[1].trim();***

***int hexValue2 = Integer.parseInt(hexValue[1]);***

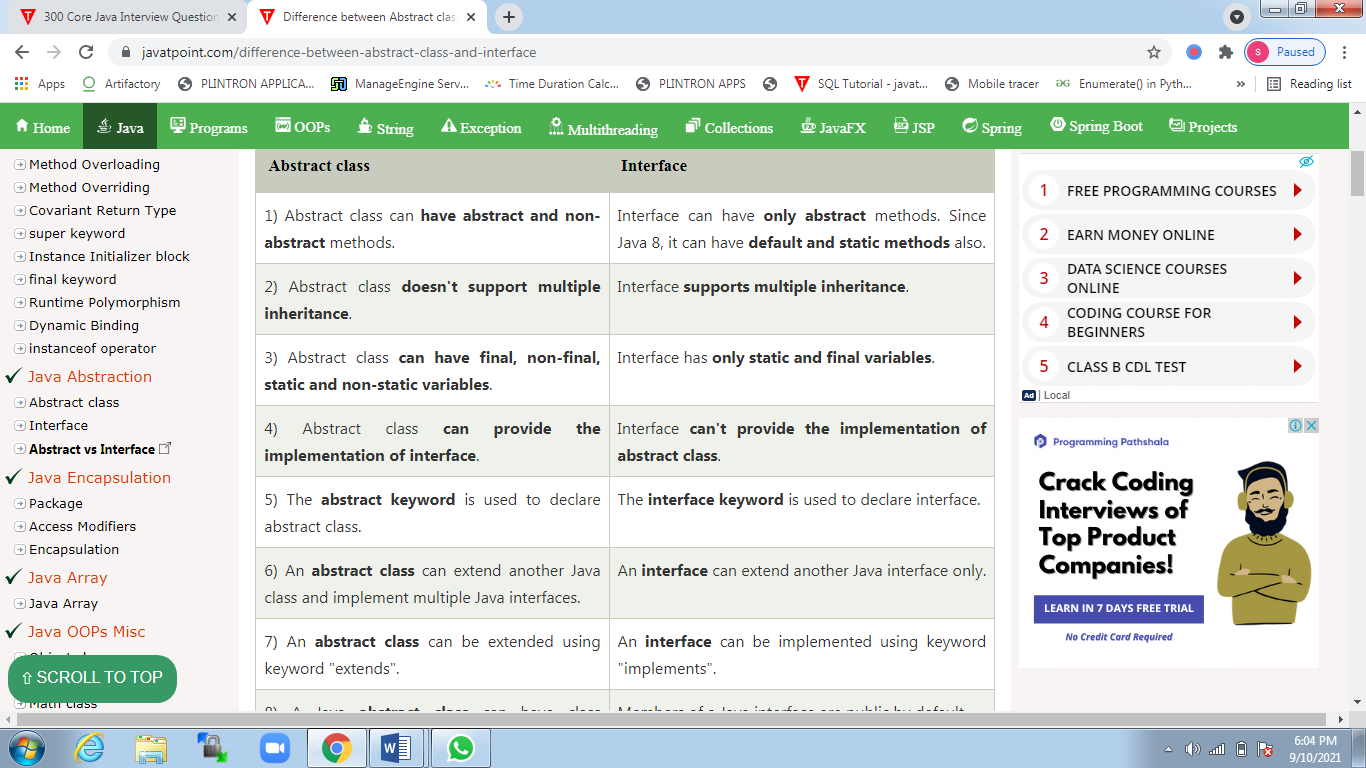
***hexValue[2] = hexValue[2].trim();***

***int hexValue3 = Integer.parseInt(hexValue[2]);***

***String actualColor = String.format(“#%02x%02x%02x”, hexValue1, hexValue2, hexValue3);***



|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |



[Blog](https://rapidapi.com/blog/) > [The Dev Room](https://rapidapi.com/blog/category/the-dev-room/) > **What’s the Difference between PUT vs PATCH?**

# What’s the Difference between PUT vs PATCH?

Last Updated on April 16, 2021 by [RapidAPI Staff](https://rapidapi.com/blog/author/bwu/) [5 Comments](https://rapidapi.com/blog/put-vs-patch/#comments)

**Table of Contents**[[show](https://rapidapi.com/blog/put-vs-patch/?utm_source=google&utm_medium=cpc&utm_campaign=DSA&gclid=Cj0KCQjw-4SLBhCVARIsACrhWLVOSktVCOj0-N1JK15Zwro2L74jtyXZKnFaZdcF704qZBd1CN6Vsl4aAmAEEALw_wcB)]

## **PUT vs PATCH**

When learning web development and HTTP specification, it is not unlikely to find yourself getting confused about the type of verb to use, and when to use it. With most applications on the internet being [CRUD (create, read/retrieve, updates, delete)](https://rapidapi.com/blog/api-glossary/crud/), developers must learn how to match HTTP verbs to these actions. Typically, the verbs and actions are matched as follows:

* [POST](https://rapidapi.com/blog/api-glossary/post/) – Create
* GET – Read/Retrieve
* PUT/[PATCH](https://rapidapi.com/blog/api-glossary/patch/) – Update
* DELETE – Delete

From this mapping, it is not surprising that most people think that PUT and PATCH are allies that do the same thing. However, the reality is far more complex, especially when it comes to overlapping functionality and other complications. Actually, PUT and PATCH might be doing the same thing of updating a resource at a location, but they do it differently. Therefore, to understand more about these verbs, let’s dive deep into HTTP specification and identify the subtle differences between the two.

[Browse the Best Free APIs List](https://rapidapi.com/collection/list-of-free-apis)

## **What is PUT?**

PUT is a method of modifying resource where the client sends data that updates the entire resource. It is used to set an entity’s information completely. PUT is similar to POST in that it can create resources, but it does so when there is a defined URI. PUT overwrites the entire entity if it already exists, and creates a new resource if it doesn’t exist.

For example, when you want to change the first name of a person in a database, you need to send the entire resource when making a PUT request.

**{**“first": "John", "last": "Stone”**}**

To make a PUT request, you need to send the two parameters; the first and the last name.

## **What is PATCH?**

Unlike PUT, PATCH applies a partial update to the resource.

This means that you are only required to send the data that you want to update, and it won’t affect or change anything else. So if you want to update the first name on a database, you will only be required to send the first parameter; the first name.