Relevantz

Engineering. And Trust.

Java 11 Features

INTRODUCTION OF JAVA 11 FEATURES

JAVA 11

- Java is a high-level programming language developed by James Gosling in 1995.
- Java is a very popular language for Android applications. Even the Android operating system itself is written in Java.
- Its syntax is simple, clean, and easy to understand so it is quite popular among developers.
- It has released many versions over the year. Java 8 and Java 11 are the versions of Java released by Oracle in 2014 and 2018 respectively, with their different functionalities.
- Java 11 is an open-source implementation of Java platform version 11, released in the year 2018.
- It comes with new features to provide more functionality.
- Many features like Appletviewer, AWT utility class, Bundled Fonts, JavaFX modules, etc are removed from Java 11, for the better functioning of the system.

Features of Java 11

- Using a new Java launcher, a single Java source code file can now be launched.
- A contemporary HTTP client, Unicode 10, nest-based access control, and more are all included in Java 11.
- Appletviewer, AWT utility class, Bundled Fonts are all removed from Java 11, while JavaFX is splitted and then separated from the JDK.
- It enables customers to receive updates for at least eight years in this version.
- A new experimental garbage collector is added.

Let us learn about them and in what aspects they are different from each other.

Difference between JAVA 8 & JAVA 11

	JAVA 8	JAVA 11
Appletviewer	The appletviewer tool is available in Java 8.	The appletviewer tool is not available in Java 11.
String Techniques	It has fewer string methods.	Several new methods of String such as isBlank(), lines(), repeat(n), and many more.
Lambda Expressions	In lambda expressions, no special variables are used.	In lambda expressions, var variables are used.
Java Deployment Technologies	This technology is available in this version.	This technology is removed from this version.
Patterns	Pattern recognition is not possible.	With the help of the asMatchPredicate() method, pattern recognition is possible
Garbage Collection	It has less garbage collector memory	It has a better garbage collection system

Difference between JAVA 8 & JAVA 11

TLS Version	It uses TLS 1.2 version.	It uses TLS 1.3 version.
Security	It is less secure when compared with JAVA 11.	In this version, the applications are quite secure than the applications in JAVA 8.
Modularity	The Modularity feature is not available in Java 8.	The Modularity feature is available in Java 11.
Var Keyword	Var keyword is not available.	Var keyword is available in Java 11, which is termed as a developer-friendly keyword.
AWTUtilities	Though the AWTUtilities class is available, it is not recommended.	The AWTUtilities class is not available.
JMC and JavaFX	These both are available in the Oracle JDK.	These both are removed from the Oracle JDK.
Performance	In JAVA 8, there is no suitable method to work with files.	In JAVA 11, there are various methods to work with the file such as writeString(), readString(), and isSameFile().

Compile Free Launch

Compile Free Launch

- From Java 11, Java provides flexibility to run Java code without compilation.
- We can execute Java code in a single step.
- By using single line command,

Syntax:

Java filename.java

Compile Free Launch

Old version of JAVA:

```
D:\>cd java11

D:\Java11>javac Sample.java

D:\Java11>java Sample
Hello World!

D:\Java11>
```

New version of JAVA 11:

```
D:\Java11>java Sample.java
Hello World!

D:\Java11>
```

File Methods

File Methods

- Java 11 introduced some new methods in the Files class and these methods provide better handling.
- These methods are:

    1) Files.writeString(path, string, options)

    2) Files.readString(filePath)

readString()

- This method is used to read all content from a file into a string.

Example:

```
public class ReadString {
	public static void main(String[] args) throws IOException {
		try {
			Path path = Paths.get("D://Java11//sample.txt");
			String data = Files.readString(path);
			System.out.println(data);
		} catch (IOException e) {
			System.out.println(e.getMessage());
		}
	}
}
```

writeString()

- This method is used to write a string to a file.

Example:

```
public class WriteString {
	public static void main(String[] args) {
		// Code to specify the file path
		Path file1 = Paths.get("D://", "Java11", "sample.txt");
		try {
			Files.writeString(file1, "\nHello from Batch 1", StandardOpenOption.APPEND);
			System.out.println("String Added Successfully!");
		} catch (IOException e) {
		}
	}
}
```

writeString() & readString()

Example:

```
public class ReadWrite {
	public static void main(String[] args) {
		// Code to specify the file path
		Path file1 = Paths.get("D://", "Java11", "sample.txt");
		try {
			Files.writeString(file1, "Hello everyone, Welcome to JAVA 11",
StandardOpenOption.WRITE);

			String value = Files.readString(file1);
			System.out.println(value);
		} catch (IOException e) {
		}
	}
}
```

Lambda Expression

What is Lambda Expression?

- Lambda expressions are the same in Java 11 as they were in previous versions of the language, having been introduced in Java 8.
- Lambda expressions are most commonly used with functional interfaces, which have a single abstract method.
- They consist of parameter list, arrow (->), and a body that defines the behavior of the lambda.
- Lambda expressions help reduce the verbosity of code, making it more readable and maintainable.

- Lambda expressions are a powerful tool for functional programming in Java, enabling developers to write more expressive and efficient code.
- A lambda expression is a short block of code which takes in parameters and returns a value.
- Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

Example

Use a lambda expression in the ArrayList's forEach() method to print every item in the list:

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();
    numbers.add(2);
    numbers.add(9);
    numbers.add(23);
    numbers.add(1);
    numbers.forEach( (n) -> { System.out.println(n); } );
  }

   }
```

Output

```
Lambda Expression:
2
9
23
1
```

Var keyword in Lambda expressions

var keyword in lambda expressions

- The var keyword was added in Java 10.
- Java 11 var is a keyword that was used while developing code in java language.
- We are using type inference in the var keyword which detects the variable data type based on the surrounding context.
- We are using the var keyword in java for declaring the local variable.
- We can also use the var keyword with lambda expression for avoiding the variable name, otherwise, it will conflict with the variable.

Declare variables with type in a lambda expression:

```
StringOp s = (String left,String  right) -> left + right;
```

Similarly, We Can Declare Without Type:

```
StringOp s = (left, right) -> left + right;
```

Example

```
import java.io.*;

interface StringOper {
    String concat(String left, String right);
}
class GFG {
    public static void main(String[] args)
    {
        // using var keyword
        StringOper s  = (var left, var right) -> left + right;
        String op = s.concat("Hello", " World");
        System.out.println(op);
    }
}
```

Output

Hello World

Rules To Follow for Var Keyword

- Lambda Despite there being only one parameter, var-declared parameters need to be surrounded in parenthesis ().
- It is not permitted to combine var with other non-var parameters. It can only var to var.
- It is likewise prohibited to use var for one parameter while skipping another.

1. (var first,var second) ➔ first + second; – valid and works.
2. (var first, second) ➔ first + second; – Invalid and throws an exception like Invalid lambda parameter declaration,Invalid AssignmentOperator

New String Methods

String Class

- Java 11 (JDK 11) added a few new **methods** in the String class.
- These methods will **reduce the complexity** of the program and **improve code readability and performance**.
- **String class** that presents in ***java.lang*** package.

String Methods

isBlank()

lines()

repeat()

strip()

stripLeading()

stripTrailing()

These all are Instance methods

isBlank() Method

- This method is used to check whether the **string is blank or not**.
- It **returns true,** if the **string is empty or contains only white space** codepoints, otherwise false.

```
public static void main(String[] args) {
        String s1 = "";
        String s2 = " ";
        String s3 = "String";
        System.out.println("s1 is blank ? : " + s1.isBlank());
        System.out.println("s2 is blank ? : " + s2.isBlank());
        System.out.println("s3 is blank ? : " + s3.isBlank());
    }
```

Output

s1 is blank ? : true
s2 is blank ? : true
s3 is blank ? : false

lines() Method

Line Terminator : \n \r

- This method **splits a string** using line terminators and **returns a stream of lines**.

```
public static void main(String[] args) {
        String str = "Hai this is java program\nthis is string lines()
method\rjava.lang.String.lines()";
        //for each with lambda
        Stream<String> strs=str.lines();
        strs.forEach(st->System.out.println(st));
        System.out.println("\n");
        //with method reference
        Stream<String> strAns = str.lines();
        strAns.forEach(System.out::println);
}
```

Output

Hai this is java program
this is string lines() method
java.lang.String.lines()

repeat() Method

- It **repeats the string ‘n’ times** (concatenate the string) and returns a string composed of this string repeated ‘n’ times.
- It will throw IllegalArgumentException if the count is *negative*.

```
public static void main(String[] args) {
        String str1 = "Java Program";
        String result1 = str1.repeat(5);
        System.out.println(result1);
    }
```

Output

Java ProgramJava ProgramJava ProgramJava ProgramJava Program

strip() Method

- It **removes leading and trailing spaces** for a given string in java11.
- Before java11, trim() method used to remove the spaces for a string.
- strip method works with Unicode characters, trim does not work with Unicode chars.
- It does not remove the spaces in middle.

```
public static void main(String[] args) {
        String st = " \t This is Java String strip()
method \u2005";
        System.out.println(st);
        System.out.println(st.strip());
}
```

Output

```
    This is Java String strip() method
This is Java String strip() method
```

stripLeading() Method

- It method returns a string whose value is this string, with all **leading or beginning spaces white space removed.**

```
public static void main(String[] args) {
        String st = "\t This is Java String
stripLeading() method\t";
        System.out.println(st);
        System.out.println(st.stripLeading());
         //last space not removed
}
```

Output

```
        This is Java String stripLeading() method

This is Java String stripLeading() method
```

stripTrailing() Method

- It removes **trailing or end spaces** for a given string in java11.

```
public static void main(String[] args) {
        String st = "\t This is Java String
stripTrailing() method\t";
        System.out.println(st);
        System.out.println(st.stripTrailing());
        //first space not removed
}
```

Output

```
This is Java String strip() method
This is Java String strip() method
```

Collection to Array

Converting a Collection to an Array

- Java 11, introduces a new method in the **java.util.Collection interface** i.e. **toArray(IntFunction generator) method.**
- This method uses to **convert a collection into an array.**
- The **toArray() method** has been a part of the Collection interface since Java 1.2. But Java 11 provides a new method that **allows for more control over the returned array.** So now the toArray() method is an **overloaded method.**
- The method introduced in java 11 takes an **IntFunction** as an argument, which specifies the type of the returned array.

Array operations

- It provides some methods that can **convert a collection into an array**.
- Each class that implements the **collection interface** must provide the implementation of the method.
- Here we have two methods:
    - **toArray() method:** This method **returns the array of Objects.** It can convert the collection to an array.
    - **toArray(T[] a) method:** This method **returns the array of T type**. It can convert the collection to a **T** typed array.

toArray() Method Sample Program

```
public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Java");
        list.add("Features");
        list.add("toArray");
        String[] array = list.stream().toArray(String[]::new);
        for (String s : array) {
            System.out.println(s);
}
```

Output

Java
Features
toArray

NESTED BASED ACCESS CONTROL

Nested based access control - Java 11

- Java 11 introduced nest-based access control that allows classes to access each other's private members without the need for **bridge methods created by the compiler.**
- These methods are called **accessibility-broadening bridge methods.**
- The compiler inserts these into the **code during the program execution.**
- Java 11 allows **classes and interfaces** to be nested within each other. These nested type can be private fields, methods, and constructors.

Nested based access control - Java 11

- In this example, we are calling a **private method inside a nested class**.
- If we execute this code using earlier versions of Java than Java 11, the compiler will create a bridge method to call the private method, but using Java 11 **there is no need of a bridge method to call private members.**

Example

```
public class Main {
        private void display() {
                System.out.println("hello from private method");
        }
        class NestedMain{
                void msg() {
                        display();
                }
        }
        }
public static void main(String[] args){

                Main m = new Main();
                Main.NestedMain n = m.new NestedMain();
                n.msg();


        }
```

Output

```
hello from private method
```

Reflection API new methods

Java added some new methods to Java Reflection

- public Class<?> getNestHost()
- public boolean isNestmateOf(Class<?> c)
- public Class<?>[] getNestMembers()

JEP REMOVES SOME JAVA MODULES

JEP REMOVES SOME JAVA MODULES

- JEP stands for Java (Math) Expression Parser
- The modules were already deprecated in Java 9.
- They are now completely removed.

Removed Packages

- java.xml.ws
- java.xml.bind
- java.activation
- java.xml.ws.annotation
- java.corba
- java.transaction
- java.se.ee
- jdk.xml.ws
- jdk.xml.bind

GARBAGE COLLECTOR

Garbage Collector

Unlike the **JVM GC** which is responsible for **allocating memory and releasing it**, **Epsilon only allocates memory.**

It allocates memory for the following things: -

- Performance testing.
- Memory pressure testing.
- VM interface testing.
- Extremely short lived jobs.
- Last-drop latency improvements.
- Last-drop throughput improvements**.**

Now Elipson is good only for test environments.

Garbage Collector

- It will lead to **OutOfMemoryError** in production and crash the applications.
- The benefit of Elipson **is no memory clearance overhead.**
- **Hence it'll give an accurate test** result of performance and we can no longer GC for stopping it.
- **Note:** This is an experimental feature.

Unleashing the Power of the Http Client API

Java 11 HTTP Client API Features

- The ever-evolving landscape of Java, the release of Java 11 brought forth a treasure trove of new features and enhancements.
- Among these, the introduction of the HTTP Client API has been a game-changer, revolutionizing the way Java applications interact with the web.
- We'll delve into the intricacies of the Java 11 HTTP Client API, exploring its capabilities and how it enhances your web interactions.
- An HttpClient is a built-in protocol used to send and retrieve requests, configured through a builder, and is immutable, allowing for multiple requests.
- The preferred protocol version (HTTP/1.1 or HTTP/2).

Requests can be sent either synchronously or asynchronously:

- send(HttpRequest, BodyHandler) blocks until the request has been sent and the response has been received.
- sendAsync(HttpRequest, BodyHandler) sends the request and receives the response asynchronously. The sendAsync method returns immediately with a CompletableFuture<HttpResponse>. The CompletableFuture completes when the response becomes available. The returned CompletableFuture can be combined in different ways to declare dependencies among several asynchronous tasks.

Features:

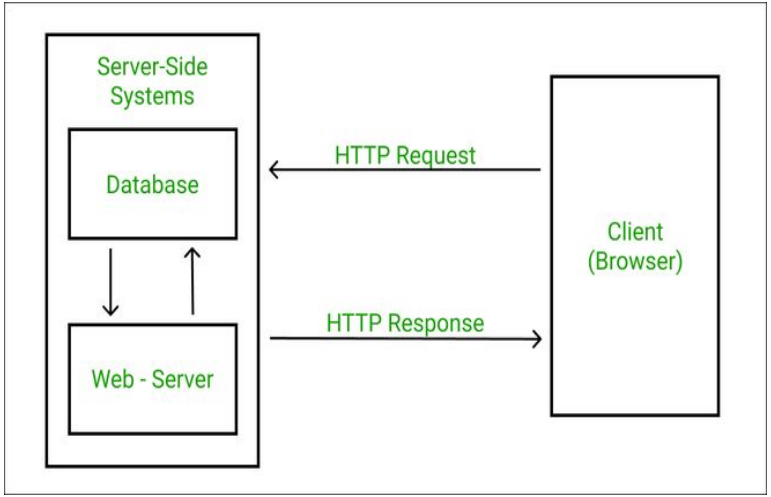
Asynchronous Request/Response Handling

Httpclient :You Gateway to the web

HttpRequest : Crafting Your Web Requests

HttpResponse : Handling Web Responses

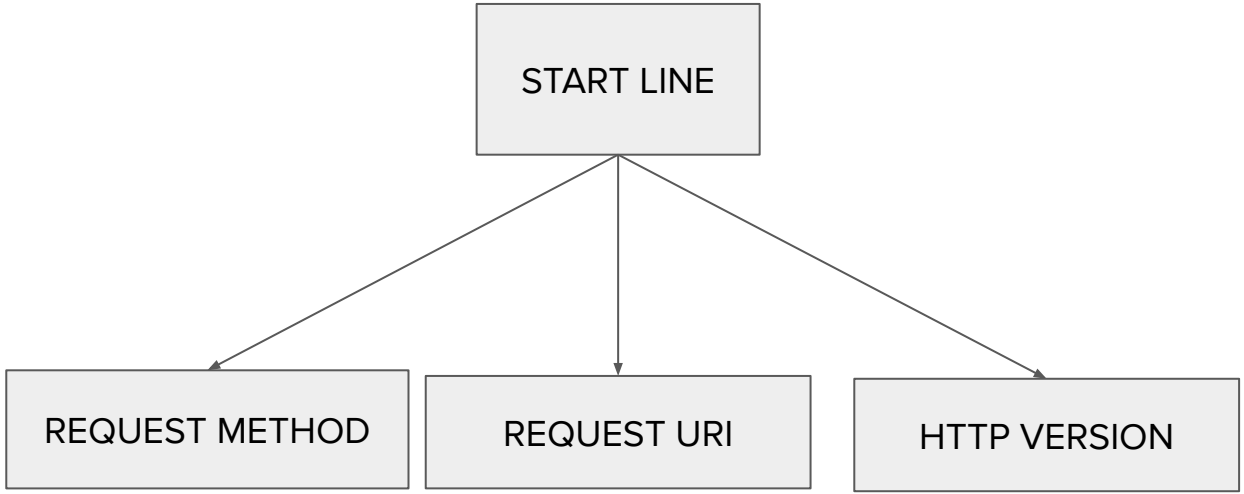
Asynchronous Requests for improved Performance



HTTP REQUEST :

STRUCTURE:

<START LINE>
<HEADER FIELDS>
<BLANK LINE>
<MESSAGE BODY>



1.Request Method:

Various method for making request

Http Methods:

1.Get - Retrieve information from specified URL
2.Post - Request the server for desired webpage
3.Head - Identical to Get Difference - Server should not return to message body response.
4.Option - Support for Specified URL. Uses Check the functionality of web server by requesting '*'.
5.Put - Uploads a representation of specified resource.
6.Delete - Deleting the specified resources.

2.Request URI:

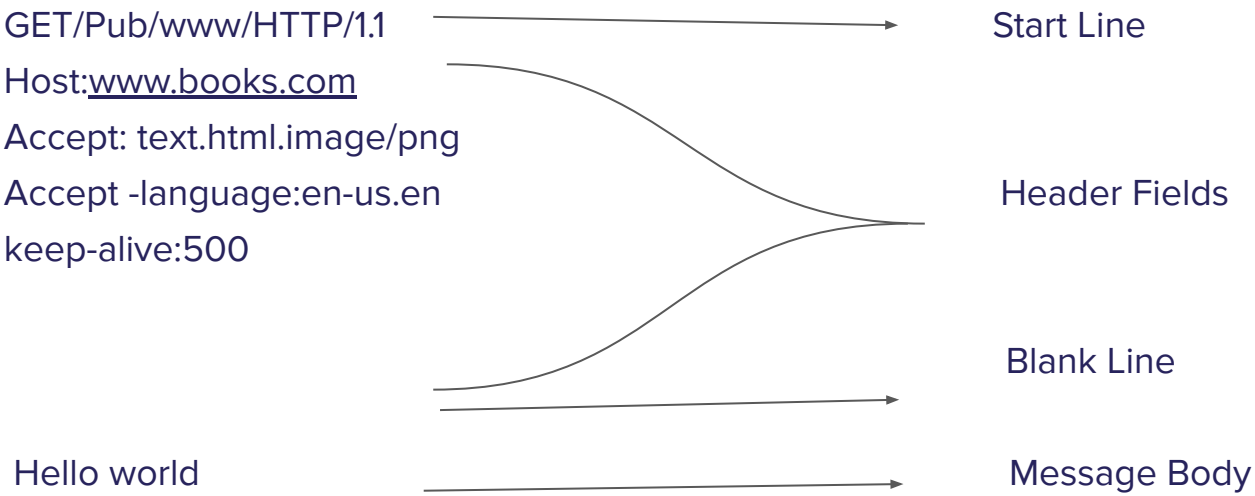
URI = URN + URL

(URI) UNIFORM RESOURCE IDENTIFIER (Identify the names)
(URN) UNIFORM RESOURCE NAME (Specified name -Person/Place) -ISBN
(URL) UNIFORM RESOURCE LOCATOR (Web address) - http:// www.website.com

3.HTTP VERSION:

- ➔ HTTP Version – HTTP/0.9
- ➔ Official – HTTP/1.1

STRUCTURE:



JEP 321:HTTP Client (Standard)

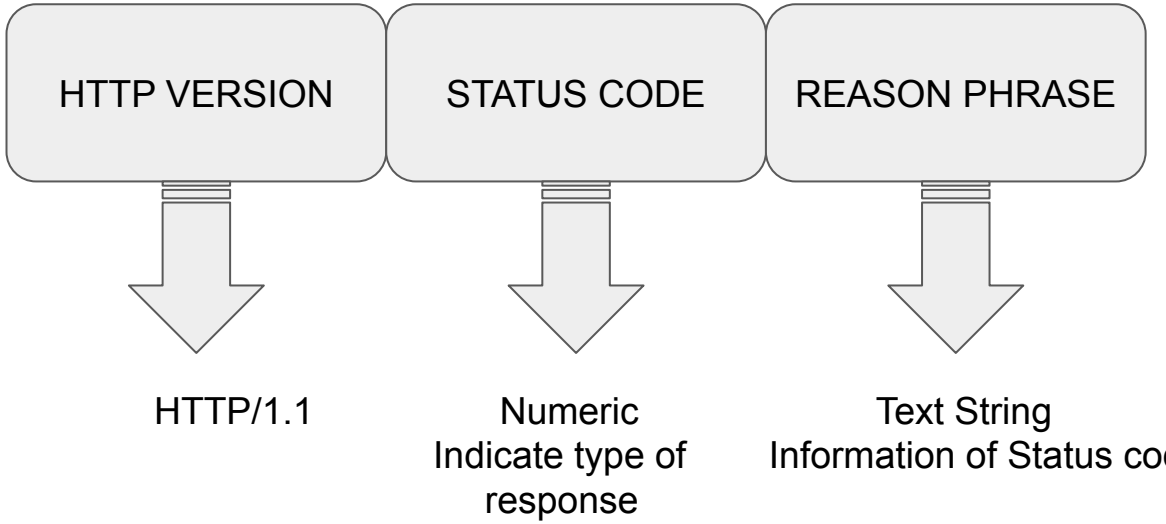
Then new HTTP client from the java.net.http package was introduced in Java 9.it has now become a standard feature in java 11.

```
HttpClient httpClient = HttpClient.newBuilder()
            .version(HttpClient.Version.HTTP_1_1)
            .connectTimeout(Duration.of Seconds(10))
            .build();

 HttpRequest request = HttpRequest.newBuilder()
            .GET()
            .uri(URI.create("https://httpbin.org/get"))
            .setHeader("User-Agent", "Java 11 HttpClient Bot")
            .build();
   HttpResponse<String> response =
     httpClient.send(request, HttpResponse.Body Handlers.ofString());

   HttpHeaders headers = response.headers();
   headers.map().forEach((k, v) -> System.out.println(k + ":" + v));
   System.out.println(response.statusCode());
   System.out.println(response.body());
```

HTTP RESPONSE Client

STRUCTURE:

<START LINE>
<HEADER FIELDS>
<BLANK LINE>
<MESSAGE BODY>



Http/1.1 200 Ok

Program:

```
package Org.java1;
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpClient.Version;
import java.net.http.HttpResponse.BodyHandlers;
public class HttpAPIDemo {
	public static void main(String[] args) {
		String uri="https://postman-echo.com/get?uname=root&pwd=root";
		HttpRequest req=HttpRequest.newBuilder()
				.uri(URI.create(uri))
				.GET()
				.version(Version.HTTP_2)
				.build();

		HttpClient client =HttpClient.newBuilder()
				.build();
				 try {
HttpResponse<String> resp=client.send(req, BodyHandlers.ofString()) ;
System.out.println(resp.statusCode());
System.out.println(resp.body());
				}catch (IOException | InterruptedException e) {
					e.printStackTrace();
				}
}}
```

Output:

```
200
{
"args": {
"uname": "root",
"pwd": "root"
},
"headers": {
"x-forwarded-proto": "https",
"x-forwarded-port": "443",
"host": "postman-echo.com",
"x-amzn-trace-id":
"Root=1-6543743d-391f6ec950f4843b25506dbd",
"user-agent": "Java-http-client/17.0.7"
},
"url":
"https://postman-echo.com/get?uname=root&pwd
=root"
}
```

Post JSON:

To post JSON data to the server using Java, you need to provide the JSON data in the HTTP POST request body and pass the "Content-Type: application/json" request header. The Content-Type request header specifies the media type for the resource in the body.

Program:

```
package Org.java1;
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.Duration;
public class PostJson {
        ////Common Formatted
                private static final HttpClient httpClient = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_2)
                .connectTimeout(Duration.of Seconds(10))
                .build();
                public static void main(String[] args) throws IOException, InterruptedException {
                // json formatted data
                String json = new StringBuilder()
                .append("{")
                .append("\"name\":\"mkyong\",")
                .append("\"notes\":\"hello\"")
                .append("}").toString();
                // add json header

                HttpRequest request = HttpRequest.newBuilder()
                .POST(Http Request.Body Publishers.of String(json))
                .uri(URI.create("https://httpbin.org/post"))
                .setHeader("User-Agent", "Java 11 HttpClient Bot") // add request header
                .header("Content-Type", "application/json")
                .build();
                HttpResponse<String> response = httpClient.send(request, HttpResponse.Body Handlers.ofString());
                // print status code
                System.out.println(response.statusCode());
                // print response body
                System.out.println(response.body());
        }
}
```

Output:

```
200
{
"args": {},
"data":
"{\"name\":\"mkyong\",\"notes\":\"hello\"}",
"files": {},
"form": {},
"headers": {
"Content-Length": "33",
"Content-Type": "application/json",
"Host": "httpbin.org",
"User-Agent": "Java 11 HttpClient Bot",
"X-Amzn-Trace-Id":
"Root=1-65437f7e-297dc62d61f82ae87616
cfc0"
},
"json": {
"name": "mkyong",
"notes": "hello"
},
"origin": "118.185.214.105",
"url": "https://httpbin.org/post"
}
```

Optional Enhancements in Java 11

Java 11 introduced new method to Optional class as isEmpty() to check if value is present. isEmpty() returns false if value is present otherwise true. It can be used as an alternative of isPresent() method which often needs to negate to check if value is not present.

```
import java.util.Optional;

public class APITester {
   public static void main(String[] args) {
      String name = null;


System.out.println(!Optional.ofNullable(name).isPresent());
      System.out.println(Optional.ofNullable(name).isEmpty());

      name = "Joe";

System.out.println(!Optional.ofNullable(name).isPresent());
      System.out.println(Optional.ofNullable(name).isEmpty());
   }
}
```

Output:

```
true
true

false
false
```

Question and Answer

Relevantz
Engineering. And Trust.

THANK YOU

JAVA 11 TEAM

About Relevantz

Relevantz Technology Services Inc. has been delivering relevant technology solutions to help improve lives for 25 years. Our team of 1200+ software engineers across 5 global offices serve customers across the finance, healthcare, insurance, media, telecom, retail, and technology sectors. Learn more at **www.relevantz.com** or @relevantz