Министерство образования, культуры и исследований Республики Молдова

Технический университет Молдовы

Департамент программной инженерии и автоматики

**Отчет**

**по лабораторной работе №3**

### **Предмет: Programarea în Reţea**

**Тема:** HTTP Client

**Выполнил:** Шарафудинов Николай

**Проверила:**  Ангилоглу А.

Кишинёв 2022

**Тема:** HTTP Client

**Цель работы:** Создать клиентское приложение HTTP

**1. Как форматируется тело запроса для POST HTTP-запроса**

**HTTP** представляет протокол, позволяющий отправлять документы в Интернете. В данном случае есть две разные роли: сервер и клиент. Как правило, клиент всегда инициирует разговор, а сервер отвечает.

**HTTP**-сообщения состоят из заголовка и тела, которое часто может оставаться пустым, но обычно оно содержит данные, которые необходимо передать по сети, чтобы использовать их в соответствии с инструкциями в заголовке, который, в свою очередь, содержит метаданные, например информацию о кодировке.

**POST** является одним из многих методов запроса, поддерживаемых HTTP протоколом. Данный метод предназначен для запроса, при котором веб-сервер принимает данные, заключённые в тело сообщения, для хранения. Он часто используется для загрузки файла или представления заполненной веб-формы. В рамках POST запроса произвольное количество данных любого типа может быть отправлено на сервер в теле сообщения запроса. POST предназначен для отправки данных на сервер. Тип тела запроса указывается в заголовке **Content-Type**. POST обычно отправляется через форму HTML и приводит к изменению на сервере.

**Синтаксис:** POST /index.html  
**Пример**:   
POST / HTTP/1.1  
Host: foo.com  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 13  
say=Hi&to=Mom

**2. Как HTTP клиент узнает, какой тип контента HTTP-сервера**

Веб-серверы ожидают сообщений с запросами от клиентов, обрабатывают их, когда они приходят и отвечают веб-браузеру через сообщение с HTTP-ответом. Ответ содержит Код статуса HTTP-ответа, который показывает, был ли запрос успешным. После того как HTML-страница возвращена, она отрисовывается браузером.

Серверный компонент предоставляет функцию или услугу одному или нескольким клиентам, которые инициируют запросы на такие услуги. Клиентское программное обеспечение может взаимодействовать с серверным программным обеспечением на том же компьютере. Клиенту нужно только понять ответ, основанный на известном протоколе приложения, то есть содержание и форматирование данных для запрашиваемой услуги. Клиент отправляет запрос, а сервер возвращает ответ. Этот обмен сообщениями является примером межпроцессного взаимодействия. Чтобы еще больше формализовать обмен данными, сервер может реализовать интерфейс прикладного программирования, API, это уровень абстракции для доступа к сервису.

**3. Как клиент решает, следует ли доверять сертификату сервера**

**SSL-сертификаты** — самый распространенный тип сертификатов в Интернете. Чаще всего их устанавливают на сайтах, которые собирают персональные данные пользователей (email, пароли, данные банковских карт и др.).

Чтобы эти данные не перехватили, соединение защищается специальным протоколом **HTTPS**, шифрующим всю информацию, которой обмениваются посетитель и сайт. Для работы этого протокола как раз и нужны SSL-сертификаты. Это безопасное соединение, которое гарантирует, что информация которая передается от вашего браузера на сервер остается приватной; то есть защищенной от хакеров или любого, кто хочет украсть информацию.

Чтобы подготовить веб-сервер для обработки https-соединений, администратор должен получить и установить в систему сертификат открытого и закрытого ключа для этого веб-сервера. **Сертификат открытого ключа** подтверждает принадлежность данного открытого ключа владельцу сайта, а **закрытый ключ** используется для расшифровки сообщений от клиента. Если имя сервера не совпадает с указанным в сертификате, то пользовательские программы, например браузеры, сообщают об этом пользователю. В основном, браузеры предоставляют пользователю выбор: продолжить незащищённое соединение или прервать его.

**4. В чем основная проблема самоподписанных сертификатов?**

Когда к сайту, защищенному **самоподписанным** сертификатом, подключается пользователь, он видит предупреждение о том, что сертификат не является безопасным. На публичных сайтах и тем более в онлайн-магазинах ни в коем случае нельзя использовать самоподписанные SSL-сертификаты.

Вторая угроза самоподписанных сертификатов — это атаки. Дело в том, что такими сертификатами любят пользоваться хакеры, совершая фишинговые атаки. При фишинге пользователей перенаправляют на сайт с похожим адресом (URL). При этом в браузере обязательно появится предупреждение про небезопасный сертификат (точно такое же, как при входе на реальный сайт с самоподписанным сертификатом).

**5. Постоянное HTTP-соединение - каковы основные преимущества?**

Постоянное соединение HTTP , представляет собой идею использования одного TCP- соединения для отправки и получения нескольких HTTP-запросов / ответов, в отличие от открытия нового соединения для каждой отдельной пары запрос / ответ. Более новый протокол HTTP / 2 использует ту же идею и развивает ее, позволяя мультиплексировать несколько одновременных запросов / ответов через одно соединение. **Основные преимущества:**

— Уменьшена задержка при последующих запросах (без квитирования ).

— Снижение использования ЦП и циклов приема-передачи из-за меньшего количества новых подключений и подтверждений TLS .

— Включает конвейерную обработку запросов и ответов HTTP .

— Уменьшение перегрузки сети (меньше TCP-соединений ). Об ошибках можно сообщать без штрафа за закрытие TCP-соединения.

**6. Что такое согласование содержимого HTTP и как оно происходит?**

В HTTP **согласование контента** - это механизм используемый для отображения различных представлений ресурса по тому же URI, так чтобы клиент мог указать, что лучше подходит для пользователя. Когда клиент хочет получить ресурс, он запрашивает его, используя его URL. Сервер использует этот URL, чтобы выбрать один из возможных вариантов - каждый вариант, назывется представлением, – и возвращает этот вариант клиенту.

В согласовании на стороне сервера или упреждающем согласовании, браузер посылает несколько заголовков HTTP наряду с URL. Эти заголовки описывают предпочтения пользователя. Сервер использует их в качестве подсказок для внутреннего алгоритма, который выбирает наиболее подходящее представление ресурса, чтобы предоставить его клиенту. Реализация алгоритма в стандарт не входит и полностью зависит от сервера.

**7. Каковы типы согласования содержимого HTTP?**

Согласование, **управляемое сервером** — если выбор наилучшего представления для отклика выполнен с использованием некоторого алгоритма на сервере, такая процедура называется согласованием под управлением сервера. Согласование, управляемое сервером, предпочтительнее, когда алгоритм выбора из числа доступных представлений трудно описать агенту пользователя. Их привлекательно тогда, когда сервер хочет послать свои наилучшие предложения клиенту вместе с первым откликом (надеясь избежать задержек RTT последующих запросов — вдруг именно это предложение удовлетворит пользователя).

В случае согласования, **управляемого клиентом**, тип содержимого определяется только на стороне клиента. Для этого сервер возвращает в ответе с кодом состояния 300 (Multiple Choices) или 406 (Not Acceptable) список вариантов, среди которых пользователь выбирает подходящий. Управляемое клиентом согласование хорошо, когда содержимое различается по самым частым параметрам (например, по языку и кодировке) и используется публичный кэш.

**8. Что такое ETag в HTTP и как он работает?**

**ETag** или **entity tag** — один из служебных заголовков протокола HTTP, который может быть установлен веб-сервером в фазе формирования ответа, на полученный от клиента запрос. Содержимое заголовка ETag является идентификатором, значение которого прямо зависит от состояния загружаемого клиентом ресурса. Это закрытый идентификатор, присвоенный веб-сервером на определённую версию ресурса, найденного на URL. Если содержание ресурса для этого адреса меняется на новое, назначается и новый ETag. Использование в таком ключе ETags аналогично использованию отпечатков пальцев, можно быстро сравнить и определить, являются ли две версии ресурса одинаковыми или нет. Сравнение ETag имеет смысл только c Etag с одного и того же URL, идентификаторы, полученные из разных URL-адресов, могут быть равны, а могут быть нет, вне зависимости от ресурсов, так что их сравнение не имеет какого-либо смысла.

Заголовок HTTP ответа ETag является идентификатором специфической версии ресурса. Он позволяет более эффективно использовать кеш и сохраняет пропускную способность, позволяя серверу отправлять не весь ответ, если содержимое не изменилось. С другой стороны, если контент все-так поменялся, Etag помогает предотвратить одновременное обновление ресурса от перезаписи друг друга ("воздушная коллизия").

**9. Разница между протоколами без состояния и без состояния. К какому типу принадлежит HTTP?**

В вычислениях протокол **без сохранения состояния** — это протокол связи, в котором информация о сеансе не сохраняется получателем. Данный протокол относит каждый запрос к независимой транзакции, которая не связана с предыдущим запросом, то есть общение с сервером состоит из независимых пар запрос-ответ. Данные сеанса отправляются получателю клиентом таким образом, что каждый переданный пакет информации может быть понят изолированно, без контекстной информации из предыдущих пакетов в сеансе. Это свойство протоколов без сохранения состояния делает их идеальными для приложений большого объема, повышая производительность за счет устранения нагрузки на сервер, вызванной сохранением информации сеанса. Данный протокол не требует, чтобы сервер сохранял информацию о сеансе или статус каждого взаимодействующего партнера в течение нескольких запросов. Напротив, протокол, который требует сохранения внутреннего состояния на сервере , известен как протокол с отслеживанием состояния. Пример протокола без сохранения состояния — HTTP, это означает, что каждое сообщение запроса может быть понято в изоляции от других запросов.

**10. Ключевые преимущества HTTP/2 по сравнению с HTTP/1.1**

В качестве усовершенствований HTTP используются такие механизмы, как **контроль потоков (flow control), апгрейд (upgrade) и обработка ошибок.** Они позволяют разработчикам обеспечивать высокую производительность и устойчивость веб-приложений. Возможность отправки данных по инициативе сервера (Server Push), предоставляемая HTTP/2, позволяет серверу отдавать сразу весь контент страницы, за исключением того, что уже имеется в кэше браузера.

HTTP/2 разрабатывался с учётом взаимозаменяемости и совместимости с HTTP 1.1. Ожидается, что внедрение HTTP/2 даст толчок к дальнейшему развитию протокола.

С технической точки зрения, одной из наиболее важных особенностей, которые отличают HTTP/1.1 и HTTP/2, является двоичный уровень кадрирования, который можно рассматривать как часть прикладного уровня в стеке интернет-протоколов. В отличие от HTTP/1.1, в котором все запросы и ответы хранятся в простом текстовом формате, HTTP/2 использует двоичный уровень кадрирования для инкапсуляции всех сообщений в двоичном формате, при этом сохраняя семантику HTTP (методы, заголовки). API прикладного уровня по-прежнему создает сообщения в обычных форматах HTTP, но нижележащий уровень преобразовывает эти сообщения в двоичные. Благодаря этому веб-приложения, созданные до HTTP/2, могут продолжать работать как обычно при взаимодействии с новым протоколом.

**11. Что такое MIME-тип, из чего он состоит и для чего используется?**

Медиа тип, так же известный как Multipurpose Internet Mail Extensions или **MIME** тип, является стандартом, который описывает природу и формат документа, файла или набора байтов. Простейший MIME тип состоит из типа и подтипа — двух строк разделённых наклонной чертой (/), без использования пробелов: **тип/подтип**

Тип представляет общую категорию, в которой находится тип данных, например video или text. Подтип же строго отождествляется с отдельным типом данных, представляемых данным MIME типом. Например, для MIME типа text, подтипы могут быть plain (простой текст), html (HTML source code) или calendar (для iCalendar/.ics).

Указание MIME-типа используется в HTML обычно при передаче данных форм и вставки на страницу различных объектов. Браузеры часто используют MIME-тип (а не расширение файла), чтобы определить, как он будет обрабатывать документ.

**12. В чем разница между GET и POST?**

HTTP методы **GET** и **POST** используются для отправки данных на сервер. POST и GET запросы можно отправить на сервер с помощью любого программного обеспечения, работающего с протоколом HTTP. Запрос GET передает данные в URL в виде пар "имя-значение" (другими словами, через ссылку), а запрос POST передает данные в теле запроса (подробно показано в примерах ниже). Это различие определяет свойства методов и ситуации, подходящие для использования того или иного HTTP метода. В таблице ниже приведены основные свойства и отличия GET и POST методов.



**13. В чем разница между PUT и POST?**

**Post** отправляет данные на сервер для создания нового объекта. Часто используется при загрузке файла или отправке веб-формы, это способ отправки данных, связанных с данным URI. Данный метод подразумевает, что Вы передаёте данные в теле запроса. POST это отправка новых данных на сервер.

**Put** похоже на POST, но используется для замены существующего объекта и предназначен как метод для «загрузки» материала в определенный URI или перезаписи того, что уже есть в этом URI. Данный метод подразумевает, что Вы передаёте всё, что нужно в URL. Тела запроса нет. PUT вносит изменения в уже имеющуюся на сервере информацию.

**14. Что такое идемпотентные методы в HTTP и какова их цель.**

Метод HTTP является **идемпотентным**, если повторный идентичный запрос, сделанный один или несколько раз подряд, имеет один и тот же эффект, не изменяющий состояние сервера. То есть, идемпотентный метод не должен иметь никаких побочных эффектов, кроме сбора статистики или подобных операций. Корректно реализованные методы GET, HEAD, PUT и DELETE идемпотентны, но не метод POST. Также все безопасные методы являются идемпотентными.

**Для идемпотентности** нужно рассматривать только изменение фактического внутреннего состояния сервера, а возвращаемые запросами коды статуса могут отличаться: первый вызов DELETE вернёт код 200, в то время как последующие вызовы вернут код 404.

**15. Как идентифицируются ресурсы в протоколе HTTP?**

Объектом HTTP запроса называется ресурс, чья природа может быть разной: фото, документ, или что-либо ещё. Обычно, чтобы описать конкретный ресурс и его местоположение, используется всего один URL. Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное. Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т. д. (в частности, для этого используется HTTP-заголовок). Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

Адрес http://example.com - это то, что мы называем URL-адресом - единым локатором ресурсов. Он представляет конкретный ресурс в Интернете.

**16. Какие существуют безопасные и небезопасные методы в HTTP?**

HTTPS является **безопасным** расширением HTTP и включает установление общего секретного ключа между клиентом и сервером, гарантируя, что мы общаемся с нужной стороной, и шифруем сообщения, которые обмениваются общим секретным ключом (подробнее об этом позже). Пока HTTPS был нацелен на повышение безопасности протокола HTTP, H2 был нацелен на обеспечение высокой скорости.

Метод HTTP является безопасным, если он не меняет состояние сервера. Другими словами, безопасный метод проводит операции "только чтение" (read-only). Несколько следующих методов HTTP безопасные: GET, HEAD или OPTIONS. Все безопасные методы являются также идемпотентными, как и некоторые другие, но при этом небезопасные, такие как PUT или DELETE. Правильная реализация безопасного метода - это ответственность серверного приложения, потому что сам веб-сервер, будь то Apache, nginx, IIS это соблюсти не сможет.

**17. Для чего нужен cURL?**

**Curl** — это сокращение от Client URL. Утилита, предназначенная для проверки подключения к URL-адресам. Это такая кроссплатформенная служебная программа командной строки, которая позволяет взаимодействовать с множеством различных серверов по множеству различных протоколов с синтаксисом URL. cURL может автоматизировать передачу файлов или последовательность таких операций, а также поддерживает протоколы: FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, LDAP, а также POP3, IMAP и SMTP. Также cURL поддерживает сертификаты HTTPS, методы HTTP POST, HTTP PUT, загрузку на FTP, загрузку через формы HTTP.

**18. Для чего нужен HTTP-прокси?**

Proxy сервер — это промежуточный компьютер, который является посредником между компьютером и интернетом. Это служба в компьютерных сетях, позволяющая клиентам выполнять косвенные запросы к другим сетевым службам. Сначала клиент подключается к такому серверу и запрашивает у него какой-либо ресурс, расположенный на другом сервере в сети\интернете. Затем сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кеша.

**19. Разница между аутентификацией и авторизацией**

**Аутентификация** — процедура проверки подлинности, например проверка подлинности пользователя путем сравнения введенного им пароля с паролем, сохраненным в базе данных. **Авторизация** — предоставление определенному лицу или группе лиц прав на выполнение определенных действий.

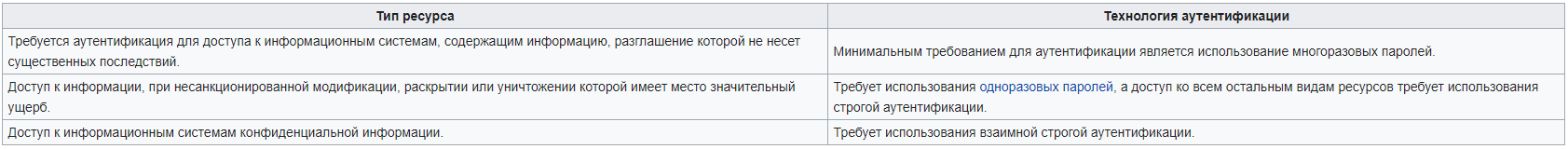
Аутентификация без предварительной идентификации лишена смысла — пока система не поймет, подлинность чего же надо проверять, совершенно бессмысленно начинать проверку. Для начала надо представиться.

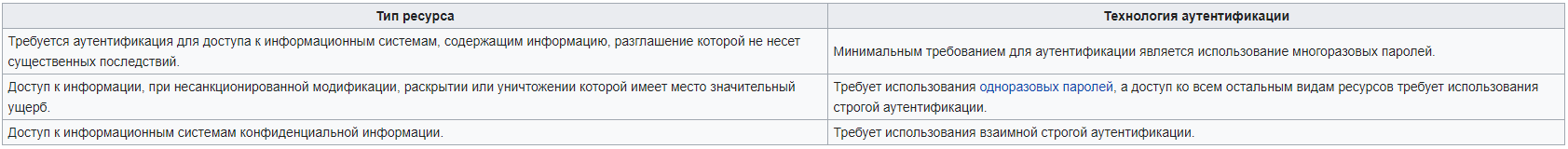
Идентификация без аутентификации — это просто глупо. Потому что мало ли кто ввел существующий в системе логин! Системе обязательно надо удостовериться, что этот кто-то знает еще и пароль. Но пароль могли подсмотреть или подобрать, поэтому лучше подстраховаться и спросить что-то дополнительное, что может быть известно только данному пользователю: например, одноразовый код для подтверждения входа.

А вот авторизация без идентификации и тем более аутентификации очень даже возможна. Например, в Google Документах можно публиковать документы так, чтобы они были доступны вообще кому угодно. В этом случае вы как владелец файла увидите сверху надпись, гласящую, что его читает неопознанный енот. Несмотря на то, что енот совершенно неопознанный, система его все же авторизовала — то есть выдала право прочитать этот документ.

**20. Каковы методы аутентификации HTTP?**

В зависимости от важности ресурса, для доступа к нему могут применяться разные методы аутентификации:





**20. Файлы cookie HTTP - для чего они нужны?**

HTTP cookie (web cookie, cookie браузера) - это небольшой фрагмент данных, отправляемый сервером на браузер пользователя, который тот может сохранить и отсылать обратно с новым запросом к данному серверу. Это, в частности, позволяет узнать, с одного ли браузера пришли оба запроса (например, для аутентификации пользователя). Они запоминают информацию о состоянии для протокола HTTP, который сам по себе этого делать не умеет. Cookie используются, главным образом, для:

**—** Управления сеансом (логины, корзины для виртуальных покупок)  
**—** Персонализации (пользовательские предпочтения)  
**—** Мониторинга (отслеживания поведения пользователя)

• Для этой лабораторной работы используйте существующие библиотеки HTTP, нет необходимости использовать Socket.

API. Кто хочет, может через розетку поступать как в первой лаборатории.

• Клиент должен делать запросы GET, POST, HEAD и OPTIONS.

• Приложение может быть консольным или графическим.

• Вы не ограничены функционалом, ресурс (веб-страница), к которому клиент будет делать HTTP-запросы, является

на выбор.

• Я рекомендую использовать частные прокси, а не бесплатные: https://proxy-seller.com

• Разработанное приложение должно иметь четко определенную логику.

Залогированный юзер может удалить других юзеров из базы данных

**Исходный код**

**GET**

import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.MalformedURLException;  
import java.net.URL;  
  
public class GET {  
 private static HttpURLConnection *connection*;  
 static BufferedReader *reader*;  
 static String *line*;  
 static StringBuffer *responseContent* = new StringBuffer();  
  
 public static void main(String[] args) throws IOException {  
  
 try {  
 URL url = new URL("https://jsonplaceholder.typicode.com/albums/1");  
 *connection* = (HttpURLConnection) url.openConnection();  
 *connection*.setRequestMethod("GET");  
  
 int status = *connection*.getResponseCode();  
 System.*out*.println(status);  
  
 String message = *connection*.getResponseMessage();  
 System.*out*.println(message);  
  
 *reader* = new BufferedReader(new InputStreamReader(*connection*.getInputStream()));  
 while ((*line* = *reader*.readLine()) != null){  
 *responseContent*.append(*line*);  
 }  
 *reader*.close();  
  
 System.*out*.println(*responseContent*.toString());  
  
 } catch (MalformedURLException e) {  
 e.printStackTrace();  
 }  
 finally {  
 *connection*.disconnect();  
 }  
 }  
}

**POST**

import java.io.BufferedReader;  
import java.io.DataOutputStream;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.nio.charset.StandardCharsets;  
  
public class POST {  
  
 private static HttpURLConnection *con*;  
  
 public static void main(String[] args) throws IOException {  
  
 var url = "https://httpbin.org/post";  
 var body = "name=Jack&occupation=programmer";  
 byte[] postData = body.getBytes(StandardCharsets.*UTF\_8*);  
  
 try {  
  
 var myurl = new URL(url);  
 *con* = (HttpURLConnection) myurl.openConnection();  
  
 *con*.setDoOutput(true);  
 *con*.setRequestMethod("POST");  
 *con*.setRequestProperty("User-Agent", "Java client");  
 *con*.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");  
  
 try (var wr = new DataOutputStream(*con*.getOutputStream())) {  
  
 wr.write(postData);  
 }  
  
 StringBuilder content;  
  
 try (var br = new BufferedReader(  
 new InputStreamReader(*con*.getInputStream()))) {  
  
 String line;  
 content = new StringBuilder();  
  
 while ((line = br.readLine()) != null) {  
 content.append(line);  
 content.append(System.*lineSeparator*());  
 }  
 }  
  
 System.*out*.println(content.toString());  
  
 } finally {  
  
 *con*.disconnect();  
 }  
 }  
}

**OPTION**

import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.MalformedURLException;  
import java.net.URL;  
  
public class OPTIONS {  
 private static HttpURLConnection *connection*;  
 static BufferedReader *reader*;  
 static String *line*;  
 static StringBuffer *responseContent* = new StringBuffer();  
  
 public static void main(String[] args) throws IOException {  
  
 try {  
 URL url = new URL("https://point.md/");  
 *connection* = (HttpURLConnection) url.openConnection();  
 *connection*.setRequestMethod("OPTIONS");  
  
 int status = *connection*.getResponseCode();  
 System.*out*.println(status);  
  
 String message = *connection*.getResponseMessage();  
 System.*out*.println(message);  
  
 *reader* = new BufferedReader(new InputStreamReader(*connection*.getInputStream()));  
 while ((*line* = *reader*.readLine()) != null){  
 *responseContent*.append(*line*);  
 }  
 *reader*.close();  
  
 System.*out*.println(*responseContent*.toString());  
  
  
 } catch (MalformedURLException e) {  
 e.printStackTrace();  
 }  
 finally {  
 *connection*.disconnect();  
 }  
  
 }  
}

**HEAD**

import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.MalformedURLException;  
import java.net.URL;  
  
public class HEAD {  
 private static HttpURLConnection *connection*;  
 static BufferedReader *reader*;  
 static String *line*;  
 static StringBuffer *responseContent* = new StringBuffer();  
  
 public static void main(String[] args) throws IOException {  
  
 try {  
 URL url = new URL("https://jsonplaceholder.typicode.com/albums/1");  
 *connection* = (HttpURLConnection) url.openConnection();  
 *connection*.setRequestMethod("HEAD");  
 *connection*.setConnectTimeout(5000);  
 *connection*.setReadTimeout(5000);  
  
 int status = *connection*.getResponseCode();  
 System.*out*.println(status);  
  
 String message = *connection*.getResponseMessage();  
 System.*out*.println(message);  
  
 *reader* = new BufferedReader(new InputStreamReader(*connection*.getInputStream()));  
 while ((*line* = *reader*.readLine()) != null){  
 *responseContent*.append(*line*);  
 }  
 *reader*.close();  
  
 System.*out*.println(*responseContent*.toString());  
  
  
 } catch (MalformedURLException e) {  
 e.printStackTrace();  
 }  
 finally {  
 *connection*.disconnect();  
 }  
  
 }  
}