*Campaña 2023- Avendaño. O*   
  
  
**Main.java:** Esta clase es el punto de entrada del programa. Su función principal es coordinar la ejecución de las clases **Busqueda, PathsTriger y Revision**. Inicia el proceso tomando el nombre original del elemento a utilizar y llama a **ModuloFiltro** que utiliza la clase **Filtro** para verificar inconsistencias en cada línea de información del archivo. Si hay inconsistencias, el programa se interrumpe. Luego, pasa a **PathsTriger**, que realiza la creación de correos electrónicos, enviando HTML para su producción en un archivo CSV. Este archivo se divide en lotes de 150,000 correos electrónicos (con posibilidad de modificación). Al finalizar, llama a la clase **Revision** para examinar cada CSV en busca de desplazamientos en las filas, identificando el número de objeto con el número ubicado en la línea del número de pago electrónico. Si se encuentra alguna inconsistencia, se emite una alerta.

**Busqueda.java**: Busca archivos en una carpeta específica con extensión .txt que contienen cierta cadena (Tipo). Devuelve el nombre del primer archivo que cumple con estas condiciones.

**PathsTriger.java**: Ejecuta la creación masiva de correos electrónicos y envía HTML para su producción en un archivo CSV. Divide el archivo en lotes de 150,000 correos electrónicos.

**Revision.java y Revisionado.java**: La clase **Revision** revisa cada archivo CSV en busca de desplazamientos en las filas. Utiliza la clase **Revisionado** para realizar la verificación. Si se encuentra alguna inconsistencia, se emite una alerta.

**Clase Filtro:**

La clase Filtro se utiliza en conjunto con el módulo de filtrado dentro de la clase **Main.** Su función principal es verificar la consistencia de cada línea de información en el archivo. A continuación, se detallan los elementos clave de la clase Filtro:

Método **verificarInconsistencias(String nombreArchivo, String line):**

Recibe como parámetros el nombre del archivo actual (nombreArchivo) y la línea actual de información (line).

Utiliza el nombre del archivo para determinar si se trata de un archivo "DT" o "CO". Luego, según el tipo de archivo, compara un carácter específico en la línea para asegurarse de que coincida con las expectativas.

Si encuentra alguna inconsistencia, devuelve false indicando que se ha detectado un problema en la línea actual.

Si no hay inconsistencias, devuelve true, indicando que la línea es consistente.

Funcionamiento:

Durante la ejecución del programa, antes de procesar cada línea de información, se llama a la clase **Filtro** para verificar que la información en la línea sea coherente con el tipo de archivo. Si se detecta alguna inconsistencia, el programa se interrumpe, evitando posibles errores en los pasos siguientes.

**Clase Revisionado:**

La clase **Revisionado** es esencial para la revisión de archivos CSV dentro de la clase **Revision**. A continuación, se detallan sus componentes clave:

**Método chequeo():**

Este método se encarga de revisar cada línea del archivo CSV en busca de posibles desplazamientos en las filas.

Utiliza varios métodos auxiliares para detectar y procesar información específica, como el objeto, el número de pago electrónico y el número en el PDF.

Compara la información obtenida para verificar la consistencia entre ellas y emite alertas si se encuentran desplazamientos o discrepancias.

**Métodos auxiliares:**

detectarYProcesar(String line): Extrae el objeto de una línea específica.

detectarYProcesar2(String line): Extrae el número de pago electrónico de una línea específica.

detectarYProcesar3(String line): Extrae el número en el PDF de una línea específica.

convertirYComparar(String str1, String str2): Convierte y compara el objeto con el número de pago electrónico, emitiendo alertas en caso de discrepancias.

**Funcionamiento:**

Durante la revisión de archivos CSV, la clase **Revisionado** procesa cada línea para extraer información clave y la compara para garantizar la coherencia entre el objeto y los números. Si se encuentra alguna discrepancia, se emite una alerta, proporcionando detalles sobre la línea y el tipo de discrepancia.

Estas clases trabajan en conjunto para asegurar la integridad de la información a lo largo del proceso y alertar sobre posibles problemas antes de avanzar a las siguientes etapas del programa.

**Verificación de Carpeta Destino:**

En aras de garantizar la integridad de la información generada, se ha implementado un módulo adicional en la clase principal (Main). Este nuevo componente tiene como función crucial verificar la existencia de archivos en la carpeta destino antes de iniciar el proceso de generación. Si la carpeta ya contiene archivos, el programa se detiene de inmediato, emitiendo una advertencia por consola: "La carpeta destino ya contiene archivos". La incorporación de esta funcionalidad proactiva contribuye a prevenir posibles conflictos al sobrescribir archivos existentes y, por ende, evita complicaciones en los módulos subsiguientes encargados de revisar y analizar la información generada. Esta medida no solo optimiza la eficiencia del proceso, sino que también fortalece la robustez y confiabilidad del sistema en su conjunto.

Sobre el nuevo HASH:   
  
  
el módulo de hash quedo d ela siguiente manera:

**public** **class** HashDTO {

**private** String hash = "";

**public** HashDTO() {}

**public** String getHash() {

**return** hash;

}

**public** **void** setHash(String hash) {

**this**.hash = hash;

}

}

**public** String hashUrl(String impuesto, String objeto, String planta, String periodo) {

// UNA URL A LA VEZ-DEFINIR CUAL: (HASH)

// String BASE\_URL = "https://app.arba.gov.ar";

String BASE\_URL = "https://clapps.test.arba.gov.ar";

String path\_hash = "/Hash/Services/ObtenerHashPorObjetoParaPDF?impuesto=" + impuesto + "&objeto=" + objeto + "&planta=" + planta + "&periodo=" + periodo;

System.***out***.println("1----->"+path\_hash);

// Construye la URL completa combinando la URL base con el path\_hash

String urlObtenerHash = BASE\_URL + path\_hash;

// Realiza la solicitud HTTP y obtiene el JSON

String json;

**try** {

json = **this**.peticionGet(urlObtenerHash);

Gson gson = **new** Gson();

HashDTO jsonObj = gson.fromJson(json, HashDTO.**class**);

**return** jsonObj.getHash();

} **catch** (ParseException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** "";

}

**public String peticionGet(String url) throws ParseException, IOException {**

**CloseableHttpClient httpClient = null;**

**CloseableHttpResponse res = null;**

**HttpEntity entity = null;**

**String jsonStr = null;**

**HttpGet httpGet = new HttpGet(url);**

**httpClient = HttpClients.*createDefault*();**

**res = httpClient.execute(httpGet);**

**Entity = res.getEntity();**

**JsonStr = EntityUtils.*toString* (entity);**

**return jsonStr;**

**}**

La función PeticionGet es la que hace finalmente el llamado al servicio y trabaja de este modo:

***CloseableHttpClient*** httpClient ***= null;***

***CloseableHttpResponse*** res ***= null;***

***HttpEntity*** entity ***= null;***

***Sting*** jsonStr ***= null;***

Se declaran varias variables que se utilizarán más adelante en el método. ***httpClient,*** ***res y entity*** son objetos de tipo *CloseableHttpClient, CloseableHttpResponse y HttpEntity,* respectivamente, que se inicializan con valor null.

***HttpGet httpGet = new HttpGet(url);***

Se crea un objeto **HttpGet** pasándole como argumento la URL que se recibió como parámetro del método.

***httpClient = HttpClients.createDefault();***

Se crea un cliente HTTP utilizando el método estático ***createDefault()*** de la clase ***HttpClients.*** Este cliente será utilizado para realizar la solicitud HTTP.

***res = httpClient.execute(httpGet);***

Se realiza la solicitud HTTP GET utilizando el cliente HTTP creado anteriormente. El resultado de esta solicitud se guarda en el objeto res de tipo ***CloseableHttpResponse***.

***entity = res.getEntity();***

Se obtiene la entidad de la respuesta HTTP (HttpEntity) utilizando el método getEntity() del objeto ***res***.

***jsonStr = EntityUtils.toString(entity);***

Se convierte la entidad HTTP en una cadena utilizando el método toString() de la clase EntityUtils.

Finalmente, se retorna la cadena resultante, - **return jsonStr; -** que contiene el cuerpo de la respuesta HTTP en formato de cadena (normalmente en formato JSON).

En resumen, este método realiza una solicitud HTTP GET a una URL dada, obtiene la respuesta, la convierte en una cadena y la retorna para su uso posterior.   
  
  
**Original:**   
  
  
>https://clapps.test.arba.gov.ar/Hash/Services/ObtenerHashPorObjetoParaPDF?impuesto=1&objeto=AF439BV&planta=&periodo=202305

**Resultado:**  
  
"hash":"01caa466d92f15c961b95b4c60d8e524cbN7e3f488bbabbd6c481961c6944800076AF439BV"

luego en otro modulo se le suma HTML :   
  
  
String link = "https://app.arba.gov.ar/pdfDeuda/emisiones/ARBADeuda/" + hash;

;

**(…)**

medioPago = "<a href=\"" + link + "\">AQUÍ</a>";

**resultado final :**   
  
 *href="https://app.arba.gov.ar/pdfDeuda/emisiones/ARBADeuda/01caa466d92f15c961b95b4c60d8e524cbN7e3f488bbabbd6c481961c6944800076AF439BV">Ingresar</a>*  
  
  
en el código embebido HTML:

