**Java 8 Stream samples in FDC**

**Java 8 AWS SDK samples in FDC**

List<String> firmSids = firmUserList.stream().map(UserAssociatedFirms::getFrmAccountId).collect(Collectors.toList());

In above code from a list of data we are isolating only firm account id as a separate list.

activeHandbookRefs.stream()

.filter((activeHandbookRef) -> !existingHandbookRefs.contains(activeHandbookRef.getHandbookReference()))

.forEach((activeHandbookRef) -> {

DataItem dataItem = new DataItem();

dataItem.setHandbookRef(activeHandbookRef.getHandbookReference());

dataItem.setDataItemName(activeHandbookRef.getDataItemName());

dataItem.setPrivileges(new Privilege());

dataItems.add(dataItem);

});

In above code we are comparing the existing handbook reference data with current handbook reference and storing only unique values.

But here we are using the stream to process and set the data to a setter instead of converting it to another new collection

map.keySet().stream().filter(s -> s.startsWith("parameter.store"))

.forEach(s -> parameterStoreMap.put(env.getProperty(s), env.getProperty(env.getProperty(s))));

parameterStoreMap.keySet().forEach(LOGGER::info);

in the above we code we are using the key set to retrieve the data with respect to that key and generate a key value pair of the properties.

**int** number = (**int**) auth.getDataItemPrivileges().stream().filter(dataItemPrivilege -> roleList.contains(dataItemPrivilege.getDataItemRoleDesc())).count();

Here count of roles are obtained from the list to perform validation in case of roles count is 0.

Set<VersionDataRequest> versionDataRequestsSet = new HashSet<VersionDataRequest>();

for (SubmitDataRequest validatereq : validateDataRequests) {

VersionDataRequest versionDataRequest = new VersionDataRequest();

versionDataRequest.setHandBookReference(validatereq.getHandbookReference());

versionDataRequest.setReportingBasis(validatereq.getReportingBasis());

versionDataRequest.setReportingEndDate(validatereq.getReportingEndDate());

versionDataRequestsSet.add(versionDataRequest);

}

versionDataRequests = versionDataRequestsSet.stream().collect(Collectors.toList());

here data already collected to a set during iteration is stored into a list for future reference.

**int** number = (**int**) auth.getDataItemPrivileges().stream().filter(dataItemPrivilege -> dataItemPrivilege.getDataItemRoleDesc().equalsIgnoreCase(requiredRole)).count();

matching the list of roles with the required roles in java

ReportListRequest reportListRequest = new ReportListRequest();

List<ReportRequest> listReportRequest = genericTransactionListResponse.getGenericTransactionList().stream()

.map(genericTransaction -> new ReportRequest(userId, genericTransaction.getFirmReferenceNumber(), genericTransaction.getReportId(), "Online"))

.collect(Collectors.toList());

reportListRequest.setReportRequestList(listReportRequest);

Here cumulative data it collected as individual object for further processing.

**return Arrays.stream(ReportType.values()).anyMatch(s->s.getValue().equals(reportType));**

**no idea what it does**

Read file line by line using stream

**private static String readFileLineByLine(String filePath) {**

**StringBuilder contentBuilder = new StringBuilder();**

**try (Stream<String> stream = Files.lines(Paths.get(filePath), StandardCharsets.UTF\_8)) {**

**stream.forEach(s -> contentBuilder.append(s).append("\n"));**

**} catch (IOException e) {**

**LOGGER.error("Exception in readFileLineByLine method {} ", e);**

**}**

**return contentBuilder.toString();**

**}**

This method collects the data from the file and creates a string and stream out of it.

profileCharacteristicNames = profileCharacteristicNames.stream().distinct().collect(Collectors.toList());

to get distinct profile characteristics from the list

List<String> profileCharNameList = request.getFirmProfile().getProfileCharacteristic()

.stream().map(ProfileCharacteristic::getName).collect(Collectors.toList());

This method return list of profiles matching the current name

reportMaterialFlow = dataItemsWithRp.stream()

.filter(object -> object.getHandbookReference().equals(HandbookRefConstants.FSA047) ||

object.getHandbookReference().equals(HandbookRefConstants.FSA048)).count()>0;

returns a Boolean result based on the filtered result count

List<String> activePeriodRagNames = activePeriodRags.stream().map(Rag::getName).collect(Collectors.toList());

List of rags based on current ragname

**return** activePeriodRags.stream().filter(rag -> nfiRags.contains(rag.getName())).collect(Collectors.toList());

List of profiles based on current ragname

**private** Boolean containIgnoreCase(List<String> profiles, String profile) {

**return** profiles.stream().anyMatch(profile::equalsIgnoreCase);

}

this will check and return if the profile is matching with the stream values from profiles

//removing duplicates

List<RagActivePeriod> uniqueRagPeriods = organizedRagPeriods.stream().distinct().collect(Collectors.toList());

**ScheduleDetermination.java has several util methods for working**

List<ReportingPeriod> filtered = keyDatesByArd.stream().filter(obj -> Util.checkDateBetween(obj.getEndDate(), ssd, sed)). .collect(Collectors.toList());

Filters based on provided input has to revisit

List<LocalDate> holidayList = holidays.stream().map(day -> Util.toLocalDate(day.getDate())).collect(Collectors.toList());

**calendarDays to LocalData mapping**

**return** list.stream().anyMatch(o ->

o.getHandbookReference().equals(handbookRef) &&

o.getNFI().equals(nfiName) &&

o.getFrequencyUnit().equals(frequencyUnit) &&

o.getFrequencyValue().equals(frequencyValue) &&

o.getTimeToCompleteUnit().equals(timeToCompleteUnit) &&

o.getTimeToCompleteValue().equals(timeToCompleteValue)

);

**public** **boolean** isCreated(List<ScheduledDataItem> items, String handbookRef, String nfi) {

**return** items.stream().anyMatch(o ->

o.getHandbookReference().equals(handbookRef) &&

o.getNeedForInformation().equals(NfiMappings.getNfiNameByNfi(nfi)

)

);

Anymatch returns Boolean based on the filter

**public** **static** **boolean** checkSumTolerance(Long totalVal, List<Long> valuesToAdd,Double toleranceUnit){

List<Double> newList = valuesToAdd.stream().map(i->

{**double** val = 0.0;

**if**(**null**!=i)

val= (**double**)i;

**return** val;

}).collect(Collectors.toList());

**if**(**null**==totalVal)

totalVal=0L;

**return** checkSumTolerance(totalVal.doubleValue(),newList,toleranceUnit);

}

**MapToDouble will convert the result to Double**

sum = valuesToAdd.stream().mapToDouble(num->{

Double retVal=0.0;

**if**(**null**!=num){

retVal = num\*monetaryUnit\*exchangeRate;

**}**

**return retVal;}).sum();**

**S3RepositoryServiceImpl.java -> should go through for some S3 related operations**

**S3BucketConnectXmlToJson**

@Override

**public** InputStreamResource downloadSubmittedFile(String keyName) {

InputStreamResource inputStreamResource;

**try** {

log.info("S3 Bucket Name: {}", env.getProperty(S3\_SUBMISSION\_BUCKETNAME));

log.info("S3 key Name: {}", CommonUtility.whiteList(keyName));

S3Object s3object = s3Client.getObject(**new** GetObjectRequest(env.getProperty(S3\_SUBMISSION\_BUCKETNAME), CommonUtility.whiteList(keyName)));

S3ObjectInputStream objectInputStream = s3object.getObjectContent();

inputStreamResource = **new** InputStreamResource(objectInputStream);

log.info("S3RepositoryServiceImpl>downloadSubmittedFile File downloaded successfully from S3 bucket");

} **catch** (AmazonClientException amazonClientException) {

log.error("An AmazonClientException occurred while downloading file from s3 {}", amazonClientException);

**throw** amazonClientException;

}

**return** inputStreamResource;

}

@Trace

@Override

**public** String getFileDataAsString(String filePath) {

**return** s3Client.getObjectAsString(env.getProperty(S3\_SUBMISSION\_BUCKETNAME), filePath);

}

**// stream to collect using trim method reference and also replacing the specific value instead of another**

**public** **static** String getContentOfENUM\_FIELD\_EXPECTED(String message) {

String str = StringUtils.substringBetween(message, SchemaErrorConditions.OPEN\_BRACKET\_MARK\_CONDITION.val(), SchemaErrorConditions.CLOSED\_BRACKET\_MARK\_CONDITION.val());

String[] elements = str.split(",");

**if** (elements.length == 1) {

**return** SchemaErrorMessages.ENUM\_SINGLE\_FIELD\_INVALID.val().replace("{value}", str);

}

List<String> list = Arrays.asList(elements);

list = list.stream().map(String::trim).collect(Collectors.toList());

str = list.stream().collect(Collectors.joining("', '", "'", "'"));

**return** SchemaErrorMessages.ENUM\_FIELD\_INVALID.val() + str + ".";

}

**SchemaValidator.java -> performValidation**

**XMLConverter.java -> toJson , convertToXml, convertJsonToXML**

**return** list.stream()

.anyMatch(listItem -> comparator.compare(listItem, item) == 1

);

This returns a Boolean based on the comparision done.

**TasksExecutor -> complete.**

**XMLtoJSONConvertor**

**DcpUtilityXmltojsonApplicationTests**

balanceSum = segregationDetails.stream().filter(segregationDetail -> **null** != segregationDetail.getBalance() && segregationDetail.getBalance() > 0).

mapToLong(SegDetailInClientMoneySegregation::getBalance).sum();

Filter with multiple condition and terminate with sum.

**CMARJsonLoader**

**UnitTestHelper**

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) **return** **true**;

// if (!(o instanceof Exposures)) return false;

// Exposures exposures = (Exposures) o;

// return Objects.equals(getExposure(), exposures.getExposure()) &&

// Objects.equals(getTotal(), exposures.getTotal());

**if** (o == **null** || ArrayList.**class** != o.getClass()) **return** **false**;

ArrayList<Exposure> that = (ArrayList<Exposure>) o;

**if**(that.size()>0 && that.stream().allMatch(n -> n ==**null** || n.isEmpty()))

**return** **true**;

**else**

**return** **false**;

}

@Override

**public** **int** hashCode() {

**return** Objects.hash(getExposure(), getTotal());

}

@Override

**public** String toString() {

**return** "Exposures{" +

"exposure=" + exposure +

", total=" + total +

'}';

}

Java.util.Collections.frequency() in Java

The method is a java.util.Collections class method. It counts the frequency of the specified element in the given list. It [override the equals()](https://www.geeksforgeeks.org/overriding-equals-method-in-java/) method to perform the comparison to check if the specified Object and the Object in the list are equal or not.

**boolean** allEmpty = **this**.firmReferenceNumber.stream().allMatch(n -> n =="" || n ==**null** );

to get all empty matches we use the above

**PDFGenerationService.java**

transactionCount = genericTransactionList.stream().collect( Collectors.groupingBy(GenericTransaction::getTransactionStatus, Collectors.counting()));

**collection.counting gives the count of each group by clause**.

List<String> duplicateValues = riskTakerMap.entrySet().stream()

.filter(x -> {

**if** (x.getValue().toString().contains(RiskTaker.ALGORITHMIC\_TRADING.toString())

|| x.getValue().toString().contains(RiskTaker.BENCHMARK\_SUBMISSION\_AND\_ADMINISTRATION.toString())

|| x.getValue().toString().contains(RiskTaker.NOT\_APPLICABLE.toString())

|| x.getValue().toString().contains(RiskTaker.STANDARD\_NON\_EXECUTIVE\_DIRECTOR.toString())

|| x.getValue().toString().contains(RiskTaker.CLIENT\_DEALING\_FUNCTION.toString())

|| x.getValue().toString().contains(RiskTaker.CASS\_OVERSIGHT\_FUNCTION.toString())

|| x.getValue().toString().contains(RiskTaker.FUNCTIONS\_SUBJECT\_TO\_QUALIFICATION\_REQUIREMENTS.toString())

|| x.getValue().toString().contains(RiskTaker.MANAGERS\_OF\_CERTIFIED\_PERSONS.toString())

|| x.getValue().toString().contains(RiskTaker.MATERIAL\_RISK\_TAKERS.toString())

|| x.getValue().toString().contains(RiskTaker.SIGNIFICANT\_MANAGEMENT\_FUNCTION.toString())

|| x.getValue().toString().contains(RiskTaker.PROPRIETARY\_TRADERS.toString())) {

**return** **true**;

}

**return** **false**;

})

// several condition check and result based on boolean

sum = valuesToAdd.stream().mapToDouble(num->{

Double retVal=0.0;

**if**(**null**!=num){

retVal = num\*monetaryUnit\*exchangeRate;

}

**return** retVal;}).sum();

List<Long> frn\_2ALongList = frn2AList.stream().filter(Objects::nonNull).mapToLong(Integer::longValue).boxed().collect(Collectors.toList());

**Here we use something called Boxed() -> which boxed them to the wrapper class to work well with collect method.**

String output = IntStream.range(0, data.size()).filter(index -> {

AuthorisedFundsOversightVisits node = data.get(index);

**boolean** result = isAllRowNull.test(node);

**return** result;

}).mapToObj( index -> parentDOM.concat(Integer.toString(index))).collect(Collectors.joining(","));

This method returns a string by concatenating integers.

numberList.**removeIf**(Objects::isNull);

**return** numberList.stream().mapToLong(Long::longValue).sum();

**Arrays.removeIf(Predicate) -> removes the data from the list based on condition**

elementRef2Errorflag = validationErrors.stream()

.anyMatch((e) -> e.getElementRef().equals(REPConstants.TWO));

This returns the Boolean based on the predicate given inside anymatch

List<String> list = Stream.of(REPConstants.ELEMENT\_D,

REPConstants.ELEMENT\_E,

REPConstants.ELEMENT\_F,

REPConstants.ELEMENT\_G,

REPConstants.ELEMENT\_H,

REPConstants.ELEMENT\_I,

REPConstants.ELEMENT\_J

).collect(Collectors.toList());

Adding stream of data from constants

Map<String, List<ValidationError>> map =

validationErrorList.stream().collect(Collectors.groupingBy(x -> {

**if**(x.getElementRef().equals(REPConstants.TWO)){

**return** x.getElementRef();

}**else**{

**return** x.getElementRef().substring(0, 2);

}

}));

Groupby based on condition and collecting.

Set<String> common = firmProfileRules.stream().distinct()

.filter(requiredRoles::contains)

.collect(Collectors.toSet());

Here distinct elements have been added based on the availability of the required roles.

ifYesList.stream().map(ifYes -> ifYes.getEnhancedReportingInvestment()).collect(Collectors.toList());

**return** !enhancementReportingInvestmentList.stream()

.filter(i -> Collections.**frequency**(enhancementReportingInvestmentList, i) > 1)

.collect(Collectors.toSet()).isEmpty();

Filter based on frequency with given predicate.

Stream.of(DataItems.values()).filter(d -> d.getValue().equals(value)).findFirst();

Returns optional of first occurrence of data item.

S3ObjectSummary latestObject = objects.stream()

.sorted(**Comparator.comparing**(S3ObjectSummary::getLastModified).**reversed**())

.filter(s -> s.getKey().endsWith(Constants.JSON))

.**findFirst().orElse**(**new** S3ObjectSummary());

This method compares using last modified time and reverse the data and filters based on predicate and findFirst() or creates a new object.

// check non-null and doesn't previously exist

overrideList.stream().map(override -> String.valueOf(getFirmSid(override.getFrn())))

.forEach(strSID -> {

FirmAttributes firm = **new** FirmAttributes(strSID, ReasonModifiedEnum.OVERRIDE);

**if** (!strSID.equalsIgnoreCase("\"null\"") &&

!firmList.getFirmAttributes().contains(firm)) {

firmList.getFirmAttributes().add(firm);

}

});

Add override based on the not null check and checks if not already exists

profileSubmissions.stream().map(profile -> **new** FirmAttributes(getFirmSid(profile.getFirmReferenceNo()), ReasonModifiedEnum.SUBMISSIONS))

.filter(firm -> checkUnique(firm, firmList))

.forEach(firm -> firmList.getFirmAttributes().add(firm));

Add unique firms to the firmList.

**final** Map<Integer, List<Long>> groups = frnListRequest.getFrnList()

.stream()

.distinct()

.collect(Collectors.groupingBy(s -> **counter.getAndIncrement()** / MAX\_FIRMS\_LIMIT));

assertThat(firmSnapshots.stream().map(FirmSnapshots::getFirmID).collect(Collectors.toList()))

.**containsExactlyInAnyOrder**("firm1", "firm2", "firm3");

In Junit we use this **containsExactlyInAnyOrder** to check this contains the data in any order

filteredEmails = filteredEmails.stream().**limit**(Constants.SENT\_EMAIL\_COUNT).collect(Collectors.toList());

Limit based on count -> which actually runs on batch

filteredEmails = filteredEmails.stream().map(data ->

{

**if** (data.getActualSentTimestamp() != **null**) {

data.setActualSentTimestamp(formatDate(data.getActualSentTimestamp()));

} **else** {

data.setScheduledSendDate(data.getScheduledSendDate());

}

**return** data;

}).collect(Collectors.toList());

Setting data model object in map operation.

**if**(errorList.stream().anyMatch(error -> error.getAddFirmToGroupCheck().equalsIgnoreCase(AddFirmToGSCheck.CHECK\_3.getId()))){

**return** AddFirmToGSCheck.CHECK\_3.getErrorMessage();

}

**if**(errorList.stream().anyMatch(error -> error.getAddFirmToGroupCheck().equalsIgnoreCase(AddFirmToGSCheck.CHECK\_4.getId()))){

**return** AddFirmToGSCheck.CHECK\_4.getErrorMessage();

}

This code performs anymatch and returns error message based on the condition.

**if** (Arrays.stream(environment.getActiveProfiles()).noneMatch(

env -> (env.equalsIgnoreCase(Constants.LOCAL\_ENV))))

if it is not local Environment , we get the code executed inside

**if** (firmProfileServiceResult.getStatus().equals(Constants.SUCCESS\_STATUS)

&& strMatch.stream().anyMatch(firmWaiverServiceResult.getStatus()::equals)

&& strMatch.stream().anyMatch(firmPermissionServiceResult.getStatus()::equals)

&& strMatch.stream().anyMatch(firmProductServiceResult.getStatus()::equals)

&& strMatch.stream().anyMatch(firmInvolvementServiceResult.getStatus()::equals)

) {

firmInformation = firmProfileServiceResult.getFirmInformation();

firmInformation.setWaiverList(firmWaiverServiceResult.getWaiverList());

firmInformation.setPermissions(firmPermissionServiceResult.getPermissionList());

firmInformation.setProducts(firmProductServiceResult.getProductList());

firmInformation.setInvolvements(firmInvolvementServiceResult.getInvolvementList());

}

**Stream with anymatch where equals is used inside**

**final** String errorMessage = Objects.requireNonNull(logsList.stream()

.filter(log -> log.getLevel().equals(Level.ERROR))

.findFirst().orElse(**null**)).getFormattedMessage();

Get formatted error message.

nfiList.stream().forEach(nfi -> {

nfi.getDisplayName();

nfi.getName();

nfiCache.put(nfi.getDisplayName(), nfi.getName());

});

Fills nfiCache Map with stream values.

pdfContents = jasperUtil.createPdf(targetStream, jsonData, dataType);

map.keySet().stream().map(map::get).forEach(output -> {

LOGGER.info("from arrayList : {} ",output);

scheduleReportTransactions.add(output);

});

Fetch the data from the map and set it to the POJO.

**public** **static** Map<String, Object> flatten(Map<String, Object> map) {

**return** map.entrySet()

.stream()

.flatMap(CompareJsonUtil::flatten)

.collect(LinkedHashMap::**new**, (m, e) -> m.put("/" + e.getKey(), e.getValue()), LinkedHashMap::putAll);

}

**private** **static** Stream<Entry<String, Object>> flatten(Entry<String, Object> entry) {

**if** (entry == **null**) {

**return** Stream.empty();

}

**if** (entry.getValue() **instanceof** Map<?, ?>) {

Map<?, ?> properties = (Map<?, ?>) entry.getValue();

**return** properties.entrySet()

.stream()

.flatMap(e -> flatten(**new** SimpleEntry<>(entry.getKey() + "/" + e.getKey(), e.getValue())));

}

**if** (entry.getValue() **instanceof** List<?>) {

List<?> list = (List<?>) entry.getValue();

**return** IntStream.range(0, list.size())

.mapToObj(i -> **new** SimpleEntry<String, Object>(entry.getKey() + "/" + i, list.get(i)))

.flatMap(CompareJsonUtil::flatten);

}

**return** Stream.of(entry);

}

This needs to be understood before saving.

**restTemplate. ( whole word and case sensitive )**