import java.util.function.Function;  
import java.util.function.Predicate;  
import java.util.stream.Collectors;

import java.util.concurrent.CompletableFuture;  
import java.util.concurrent.ConcurrentHashMap;  
import java.util.concurrent.ExecutionException;

**private** **static** <T> Predicate<T> distinctByKey(Function<? **super** T, ?> keyExtractor) {

Map<Object, Boolean> seen = **new** ConcurrentHashMap<>();

**return** t -> seen.putIfAbsent(keyExtractor.apply(t), Boolean.***TRUE***) == **null**;

}

@Override

**public** List<RIMasterGetRes> getallRIMasters() {

List<RIMasterGetRes> resList = **new** ArrayList<RIMasterGetRes>();

ModelMapper mapper = **new** ModelMapper();

**try** {

List<RIMasterDetails> rimasterdetails = riMasterRepo.findByStatusOrderByRiCodeAsc("Y");

// Filter Multiple List

rimasterdetails = rimasterdetails.stream().filter(*distinctByKey*(o -> Arrays.*asList*(o.getRiCode()))).collect(Collectors.*toList*());

**for** (RIMasterDetails data : rimasterdetails) {

RIMasterGetRes res = **new** RIMasterGetRes();

res = mapper.map(data, RIMasterGetRes.**class**);

resList.add(res);

}

} **catch** (Exception e) {

e.printStackTrace();

log.info("Log Details" + e.getMessage());

**return** **null**;

}

**return** resList;

}

@Override

**public** List<RIMasterGetRes> getallRIMasterss() {

List<RIMasterGetRes> resList = **new** ArrayList<RIMasterGetRes>();

ModelMapper mapper = **new** ModelMapper();

**try** {

List<RIMasterDetails> rimasterdetails = riMasterRepo.OrderByRiCodeAsc();

// Filter Multiple List

rimasterdetails = rimasterdetails.stream().filter(*distinctByKey*(o -> Arrays.*asList*(o.getRiCode()))).collect(Collectors.*toList*());

**for** (RIMasterDetails data : rimasterdetails) {

RIMasterGetRes res = **new** RIMasterGetRes();

res = mapper.map(data, RIMasterGetRes.**class**);

resList.add(res);

}

} **catch** (Exception e) {

e.printStackTrace();

log.info("Log Details" + e.getMessage());

**return** **null**;

}

**return** resList;

}

}