```
package ch.ucc.apim.ccapi.account.service.rules;


import ch.ucc.apim.ccapi.account.basedata.AccountStatusTable;
import ch.ucc.apim.ccapi.account.basedata.WinnerAccountStatusTable;
import ch.ucc.apim.ccapi.account.data.account.model.AccountEntity;
import ch.ucc.apim.ccapi.account.data.account.repo.AccountDAO;
import ch.ucc.apim.ccapi.accounts.api.domain.AccountDetails;
import ch.ucc.apim.ccapi.accounts.api.domain.CardDetails;
import ch.ucc.apim.ccapi.accounts.api.domain.StatusReason;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.EnumUtils;
import org.kie.api.event.rule.AfterMatchFiredEvent;
import org.kie.api.event.rule.BeforeMatchFiredEvent;
import org.kie.api.event.rule.DefaultAgendaEventListener;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.springframework.stereotype.Component;


import java.math.BigInteger;
import java.util.Map;


import static
ch.ucc.apim.ccapi.account.mapping.MappingConstants.GLOBAL_ACCOUNT_STATUS_TABLE;


@Component
@RequiredArgsConstructor
@Getter
@Setter
```

```
@Slf4j
public class AccountStatusRulesEngine {

    private final KieContainer kieContainer;
    private final Map<BigInteger, AccountEntity> selectedAccountsMap;
    private final AccountStatusTable accountStatusTable;
    private final WinnerAccountStatusTable winnerAccountStatusTable;
    private KieSession kieSession;
    private AccountDetails accountDetails;
    private Map<String, AccountEntity> accountEntityMap;

    public synchronized void provideAccountStatus(AccountDAO accountDAO,
                                                  String accountId, AccountDetails accountDetails) {
        AccountEntity accountEntity = accountDAO.findAccountByAccountId(accountId);
        if (accountEntity.getAccountLevel() == null)
            accountEntity.setAccountLevel(BigInteger.valueOf(Long.parseLong("1")));
        String parentAccount = accountEntity.getParentAccountId();
        selectedAccountsMap.put(accountEntity.getAccountLevel(), accountEntity);
        if (parentAccount == null) {
            initiateSession();

        } else {
            provideAccountStatus(accountDAO, parentAccount, accountDetails);
        }
        applyRules();
        if (selectedAccountsMap.isEmpty()) {

accountDetails.setAccountStatus(AccountDetails.AccountStatusEnum.valueOf(winnerAccountStatus
Table.getAccountStatus()));
            StatusReason statusReason = new StatusReason();
            if (!winnerAccountStatusTable.getStatusReasonCode().isEmpty())
```

```
statusReason.setStatusReasonCode(StatusReason.StatusReasonCodeEnum.valueOf(winnerAccountS
tatusTable.getStatusReasonCode()));
            accountDetails.setStatusReason(statusReason);
            setAccountDetails(accountDetails);
            kieSession.dispose();
            winnerAccountStatusTable.clear();
        }
    }

    public synchronized void  provideNonLiableCardStatus(AccountDAO accountDAO, String
accountId, CardDetails cardDetails) {
        AccountEntity accountEntity = accountDAO.findAccountByAccountId(accountId);
        if (accountEntity.getAccountLevel() == null)
            accountEntity.setAccountLevel(BigInteger.valueOf(Long.parseLong("1")));
        String parentAccount = accountEntity.getParentAccountId();
        selectedAccountsMap.put(accountEntity.getAccountLevel(), accountEntity);
        if (parentAccount == null) {
            initiateSession();


        } else {
            provideNonLiableCardStatus(accountDAO, parentAccount, cardDetails);
        }
        applyRules();
        if (selectedAccountsMap.isEmpty()) {
            String cardStatus = winnerAccountStatusTable.getAccountStatus();
            CardDetails.CardStatusEnum cardStatusValue =
EnumUtils.isValidEnum(CardDetails.CardStatusEnum.class,
cardStatus)?CardDetails.CardStatusEnum.valueOf(cardStatus) : null;
            cardDetails.setCardStatus(cardStatusValue);
            cardDetails.setStatusReason(accountDetails.getStatusReason());
            kieSession.dispose();
            winnerAccountStatusTable.clear();
        }
    }


    public  void provideCardStatusEmbedded(AccountDAO accountDAO, CardDetails cardDetails) {
        if (accountDetails != null && cardDetails.getCardId().equals(accountDetails.getAccountId())) {
            AccountDetails.AccountStatusEnum accountStatusEnum = accountDetails.getAccountStatus();
            String cardStatusValue = EnumUtils.isValidEnum(CardDetails.CardStatusEnum.class,
accountStatusEnum.name()) ? accountStatusEnum.name() : null;
            cardDetails.setCardStatus(CardDetails.CardStatusEnum.valueOf(cardStatusValue));
            StatusReason statusReason = accountDetails.getStatusReason();
            cardDetails.setStatusReason(statusReason);
        } else if (accountDetails != null &&
!cardDetails.getCardId().equals(accountDetails.getAccountId())){
            provideNonLiableCardStatus(accountDAO, cardDetails.getCardId(), cardDetails);
        }
    }


    private void initiateSession() {
        kieSession = kieContainer.newKieSession();
        kieSession.setGlobal(GLOBAL_ACCOUNT_STATUS_TABLE, accountStatusTable);
        kieSession.addEventListener(new MyDefaultAgendaEventListener());
    }


    private void applyRules() {
        BigInteger index = new BigInteger(String.valueOf(selectedAccountsMap.size()));
        AccountEntity accountEntity = selectedAccountsMap.get(index);
        kieSession.insert(accountEntity);
        kieSession.fireAllRules();


    }


    private class MyDefaultAgendaEventListener extends DefaultAgendaEventListener {
        @Override
        public void afterMatchFired(AfterMatchFiredEvent event) {
            super.afterMatchFired(event);
            AccountStatusTable accountStatusTableTemp = (AccountStatusTable)
kieSession.getGlobal(GLOBAL_ACCOUNT_STATUS_TABLE);
            evaluateWinnerAccountStatus(winnerAccountStatusTable, accountStatusTableTemp);
            accountStatusTable.clear();
        }

        @Override
        public void beforeMatchFired(BeforeMatchFiredEvent event) {
            super.beforeMatchFired(event);
            BigInteger index = new BigInteger(String.valueOf(selectedAccountsMap.size()));
            selectedAccountsMap.remove(index);
        }

        private void evaluateWinnerAccountStatus(WinnerAccountStatusTable
winnerAccountStatusTable, AccountStatusTable accountStatusTable) {
            if ((winnerAccountStatusTable.getAccountLevel() > accountStatusTable.getAccountLevel() &&
(winnerAccountStatusTable.getAccountStatusPriorityIndex() <=
accountStatusTable.getAccountStatusPriorityIndex()))) {
                winnerAccountStatusTable.copy(accountStatusTable);
            } else if ((winnerAccountStatusTable.getAccountLevel() <
accountStatusTable.getAccountLevel() &&
(winnerAccountStatusTable.getAccountStatusPriorityIndex() <
accountStatusTable.getAccountStatusPriorityIndex()))) {
                winnerAccountStatusTable.copy(accountStatusTable);
            } else if (winnerAccountStatusTable.getAccountLevel() == 0) {
                winnerAccountStatusTable.copy(accountStatusTable);
            }
        }
    }
}
```

```
import static ch.ucc.apim.ccapi.account.mapping.MappingConstants.RULES_DEFINITION;


@Configuration
public class AccountStatusRulesConfiguration {


    private final KieServices kieServices = KieServices.Factory.get();



    @Bean
    public KieContainer getKieContainer() {
        KieFileSystem kieFileSystem = kieServices.newKieFileSystem();
        kieFileSystem.write(ResourceFactory.newClassPathResource(RULES_DEFINITION));
        KieBuilder kb = kieServices.newKieBuilder(kieFileSystem);
        kb.buildAll();
        KieModule kieModule = kb.getKieModule();
        return kieServices.newKieContainer(kieModule.getReleaseId());
    }

}
```