**JUNIT**

1. Create a new object if optional return empty

LoanUploadFileEntity loanUploadFileEntity = loanUploadFileRepository.getDNVVByLoanId(loanId)  
 .map(obj -> {  
 obj.setUpdatedAt(Instant.*now*());  
 return Optional.*of*(obj);  
 }).orElseGet(() -> Optional.*of*(LoanUploadFileEntity.*builder*()  
 .fileConfigId(fileConfigEntity.getUuid())  
 .loanApplicationId(loanId)  
 .folder(String.*format*("%s/%s/%s/", Constants.*SERVICE\_NAME*, DateUtils.*convertToSimpleFormat*(Date.*from*(loanApplication.getCreatedAt()), "yyyyMMdd"), loanId))  
 .fileName(Constants.*FILE\_NAME\_DNVV*)  
 .status(LoanUploadFileStatusEnum.*UPLOADED*)  
 .build())).get();

1. Test private

Method privateMethod = LoanUploadFileServiceImpl.class.getDeclaredMethod("renameDuplicateInZipBlock", ZipBlock.class);  
privateMethod.setAccessible(true);  
privateMethod.invoke(loanUploadFileService, zipBlock);

1. Mock static
2. try (MockedStatic<AuthorizationUtil> theMock = Mockito.*mockStatic*(AuthorizationUtil.class)) {  
    theMock.when(() -> AuthorizationUtil.*getEmail*())  
    .thenReturn(EMAIL);  
     
    ApplicationException exception = *assertThrows*(ApplicationException.class, () -> {  
    loanApplicationService.updateRmForLoan(LOAN\_ID, "3256b70fd39cfa577c857aaac33d8214d74afe19d7bcad0d39893f2");  
    });  
    *assertEquals*(exception.getCode().intValue(), ErrorEnum.*SERVER\_ERROR*.getCode());  
   }
3. Exception
4. ApplicationException exception = *assertThrows*(ApplicationException.class, () -> {  
    commonIncomeService.save(CommonIncomeMapper.*INSTANCE*.toModel(request));  
   });

CustomException exception = *assertThrows*(CustomException.class, () -> {  
 debitCardService.completeTranCashByQR(completeTransactionRequest);  
});  
  
*assertEquals*("ERROR.CBQ.COMPLETE\_TRAN", exception.getCode());

public static String *toString* = Stream.*of*(*values*()).map(CMSTabEnum::getCode)  
 .collect(Collectors.*joining*(", "));

1. Argument passed to when() is not a mock!

void givenValidInput\_ThenGetAuthType\_returnAuthTypeEqualD() {  
 UserInfo userInfo = new UserInfo("testUser", "password", true, true, true, true, List.*of*());  
 *when*(authentication.getPrincipal()).thenReturn(userInfo);  
 SecurityContextHolder.*setContext*(securityContext);  
 *when*(securityContext.getAuthentication()).thenReturn(authentication);  
  
 BcAuthTransService bcAuthTransServiceMocksSpy = *spy*(bcAuthTransService);  
 *doReturn*("D").when(bcAuthTransServiceMocksSpy).getAuthType(*any*(), *any*(), *any*(), *any*(), *any*(), *any*());  
 Map map = bcAuthTransServiceMocksSpy.getAuthType(*any*(), *any*(), *any*(), *any*(), *any*());  
  
 *assertEquals*("D", map.get("authType"));  
}

1. org.mockito.exceptions.misusing.UnnecessaryStubbingException: Unnecessary stubbings detected.

* MOCK KHÔNG CẦN THIẾT TRONG LÚC RUN UNITTEST

1. Spy (Partial Mock)

PayV1ServiceImpl spyPayV1ServiceImpl = *spy*(payV1ServiceImpl);  
*doNothing*().when(spyPayV1ServiceImpl).checkValidateValue(*any*(), *any*(), *any*());  
  
spyPayV1ServiceImpl.process(request);

2. Unit test method naming convention
3. @Test  
   void testUpdateTabValidationIfListFilesAreEmpty\_ShouldPublishEvent(){  
     
   }

testCalculateTotal\_WhenCartIsEmpty\_ShouldReturnZero() testLogin\_WithInvalidCredentials\_ShouldThrowException()

1. Convert JSON to Object
2. *when*(customerService.getCustomerRelationDetail(*any*())).thenReturn(JsonUtil.*convertStringToObj*(getCustomerRelationDetailResponseStr  
    , new TypeReference<List<SearchRelationResponse>>() {  
    }));
3. Test private

Method privateMethod = GeneralInfoServiceImpl.class.getDeclaredMethod("addAssetOwnersToCustomerInfo", AssetData.class, new ParameterizedTypeReference<List<ApplicationInfoToGIRequest.CusInfo>>() {  
});  
privateMethod.setAccessible(true);  
privateMethod.invoke(assetData, cusInfoList);

1. Không encode url

RequestEntity<Void> request = RequestEntity  
        .delete(URI.create(finalUrl)) // URI.create sẽ giữ nguyên URL  
        .header("X-MBX-APIKEY", apiKey)  
        .build();

1. DSAD
2. DSAD