package com.project.bos.dg.datastore.service.impl;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.JsonNode;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.databind.node.ArrayNode;  
import com.fasterxml.jackson.databind.node.ObjectNode;  
import com.project.bos.dg.datastore.constants.DocumentGeneratorEventStoreConstants;  
import com.project.bos.dg.datastore.mapper.EventMapper;  
import com.project.bos.dg.datastore.model.entity.DmEvent;  
import com.project.bos.dg.datastore.model.entity.DmEventStatus;  
import com.project.bos.dg.datastore.model.entity.DocumentServiceRequest;  
import com.project.bos.dg.datastore.model.entity.DocumentServiceRequestStatus;  
import com.project.bos.dg.datastore.model.enums.EventStatusType;  
import com.project.bos.dg.datastore.model.request.EventDataRequest;  
import com.project.bos.dg.datastore.model.request.EventRequest;  
import com.project.bos.dg.datastore.model.request.printservice.PrintDocumentsServiceRequest;  
import com.project.bos.dg.datastore.model.response.EventResponse;  
import com.project.bos.dg.datastore.repository.DmEventRepository;  
import com.project.bos.dg.datastore.repository.DmEventStatusRepository;  
import com.project.bos.dg.datastore.repository.DocumentServiceRequestRepository;  
import com.project.bos.dg.datastore.repository.DocumentServiceRequestStatusRepository;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.dao.DataAccessException;  
  
import java.util.\*;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.ArgumentMatchers.\*;  
import static org.mockito.Mockito.\*;  
  
@SpringBootTest  
class EventStoreHelperTest {  
 @InjectMocks  
 private EventStoreHelper eventStoreHelper;  
  
 @Mock  
 private EventMapper eventMapper;  
  
 @Mock  
 private DmEventRepository dmEventRepository;  
  
 @Mock  
 private DmEventStatusRepository dmEventStatusRepository;  
  
 @Mock  
 private DocumentServiceRequestRepository documentServiceRequestRepository;  
  
 @Mock  
 private DocumentServiceRequestStatusRepository documentServiceRequestStatusRepository;  
  
 @Mock  
 private ObjectMapper objectMapper;  
  
 @BeforeEach  
 //Setup Method  
 public void setup() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
 //Method -1 Testcase: 1  
 //Basic Functionality Test:  
 //Test that when a valid EventRequest and EventResponse are provided, the getEventDataResponse method performs  
 // the necessary database operations and mapping correctly.  
 //Verify that the DmEvent, DmEventStatus, and DocumentServiceRequest entities are saved appropriately.  
 @Test  
 public void testGetEventDataResponseBasicFunctionality() throws JsonProcessingException {  
 // Create a sample EventRequest and EventResponse  
 EventRequest eventRequest = new EventRequest();  
 EventResponse eventResponse = new EventResponse();  
  
 // Mock behavior for eventMapper  
 *when*(eventMapper.eventRequestToDocumentEventEntity(eventRequest)).thenReturn(new DmEvent());  
 // when(eventMapper.updateNewEventResponse(eventRequest, eventResponse)).thenReturn(eventResponse);  
 *doNothing*().when(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
  
 // Mock behavior for dmEventRepository  
 *when*(dmEventRepository.save(*any*(DmEvent.class))).thenReturn(new DmEvent());  
  
 // Mock behavior for dmEventStatusRepository  
 *when*(dmEventStatusRepository.save(*any*(DmEventStatus.class))).thenReturn(new DmEventStatus());  
  
 // Mock behavior for documentServiceRequestRepository  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
 *when*(documentServiceRequestRepository.saveAll(*anyList*())).thenReturn(documentServiceRequests);  
  
 // Call the method under test  
 Optional<EventResponse> result = eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify that the expected methods were called  
 *verify*(eventMapper).eventRequestToDocumentEventEntity(eventRequest);  
 *verify*(dmEventRepository).save(*any*(DmEvent.class));  
 *verify*(dmEventStatusRepository).save(*any*(DmEventStatus.class));  
 *verify*(documentServiceRequestRepository).saveAll(*anyList*());  
 *verify*(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
  
 // Assert the result  
 *assertSame*(eventResponse, result.orElse(null));  
 }  
 //Method- 1 Testcase:2  
 //2. Mocking Dependencies:  
 //• Test that the EventStoreHelper class correctly interacts with its dependencies (eventMapper, dmEventRepository,  
 // dmEventStatusRepository, documentServiceRequestRepository) by using Mockito to verify method calls and arguments  
 @Test  
 public void testDependencyInteraction() throws Exception {  
 // Create a sample EventRequest and EventResponse  
 EventRequest eventRequest = new EventRequest();  
 EventResponse eventResponse = new EventResponse();  
  
 // Mock behavior for eventMapper  
 *when*(eventMapper.eventRequestToDocumentEventEntity(eventRequest)).thenReturn(new DmEvent());  
 // when(eventMapper.updateNewEventResponse(eventRequest, eventResponse)).thenReturn(eventResponse);  
 *doNothing*().when(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
  
 // Mock behavior for dmEventRepository  
 *when*(dmEventRepository.save(*any*(DmEvent.class))).thenReturn(new DmEvent());  
  
 // Mock behavior for dmEventStatusRepository  
 *when*(dmEventStatusRepository.save(*any*(DmEventStatus.class))).thenReturn(new DmEventStatus());  
  
 // Mock behavior for documentServiceRequestRepository  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
 *when*(documentServiceRequestRepository.saveAll(*anyList*())).thenReturn(documentServiceRequests);  
  
 // Call the method under test  
 eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify interactions with dependencies  
 *verify*(eventMapper).eventRequestToDocumentEventEntity(eventRequest);  
 *verify*(dmEventRepository).save(*any*(DmEvent.class));  
 *verify*(dmEventStatusRepository).save(*any*(DmEventStatus.class));  
 *verify*(documentServiceRequestRepository).saveAll(*anyList*());  
 *verify*(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
  
 // Additional verifications, if needed, for method arguments or number of invocations  
 *verify*(dmEventStatusRepository).setStatus(*anyString*());  
 }  
  
 //Method-1 Testcase:3  
 //3. Empty DocumentServiceRequest List:  
 //• Test the case where getPrintFileRequest returns an empty list.  
 // Ensure that the method behaves correctly, and no additional entities are saved  
  
 @Test  
 public void testEmptyDocumentServiceRequest() throws Exception {  
 // Create a sample EventRequest and EventResponse  
 EventRequest eventRequest = new EventRequest();  
 EventResponse eventResponse = new EventResponse();  
  
 // Mock behavior for eventMapper  
 *when*(eventMapper.eventRequestToDocumentEventEntity(eventRequest)).thenReturn(new DmEvent());  
 //when(eventMapper.updateNewEventResponse(eventRequest, eventResponse)).thenReturn(eventResponse);  
 *doNothing*().when(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
  
 // Mock behavior for dmEventRepository  
 *when*(dmEventRepository.save(*any*(DmEvent.class))).thenReturn(new DmEvent());  
  
 // Mock behavior for dmEventStatusRepository  
 *when*(dmEventStatusRepository.save(*any*(DmEventStatus.class))).thenReturn(new DmEventStatus());  
  
 // Mock behavior for documentServiceRequestRepository (empty list)  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
 *when*(documentServiceRequestRepository.saveAll(*anyList*())).thenReturn(documentServiceRequests);  
  
 // Call the method under test  
 eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify interactions with dependencies  
 *verify*(eventMapper).eventRequestToDocumentEventEntity(eventRequest);  
 *verify*(dmEventRepository).save(*any*(DmEvent.class));  
 *verify*(dmEventStatusRepository).save(*any*(DmEventStatus.class));  
  
 // Verify that documentServiceRequestRepository.saveAll was not called  
 *verify*(documentServiceRequestRepository, *never*()).saveAll(*anyList*());  
  
 *verify*(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
 }  
 //Method-1 testcase : 4  
 //Non-Empty DocumentServiceRequest List:  
 // Test the case where getPrintFileRequest returns a non-empty list of DocumentServiceRequest objects.  
 // Ensure that these requests are saved correctly and their statuses are set appropriately.  
 @Test  
 public void testNonEmptyDocumentServiceRequest() throws Exception {  
 // Create a sample EventRequest and EventResponse  
 EventRequest eventRequest = new EventRequest();  
 EventResponse eventResponse = new EventResponse();  
  
 // Mock behavior for eventMapper  
 *when*(eventMapper.eventRequestToDocumentEventEntity(eventRequest)).thenReturn(new DmEvent());  
 //when(eventMapper.updateNewEventResponse(eventRequest, eventResponse)).thenReturn(eventResponse);  
 *doNothing*().when(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
  
 // Mock behavior for dmEventRepository  
 *when*(dmEventRepository.save(*any*(DmEvent.class))).thenReturn(new DmEvent());  
  
 // Mock behavior for dmEventStatusRepository  
 *when*(dmEventStatusRepository.save(*any*(DmEventStatus.class))).thenReturn(new DmEventStatus());  
  
 // Mock behavior for documentServiceRequestRepository (non-empty list)  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
 documentServiceRequests.add(new DocumentServiceRequest()); // Add sample request(s)  
 *when*(documentServiceRequestRepository.saveAll(*anyList*())).thenReturn(documentServiceRequests);  
  
 // Call the method under test  
 eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify interactions with dependencies  
 *verify*(eventMapper).eventRequestToDocumentEventEntity(eventRequest);  
 *verify*(dmEventRepository).save(*any*(DmEvent.class));  
 *verify*(dmEventStatusRepository).save(*any*(DmEventStatus.class));  
  
 // Verify that documentServiceRequestRepository.saveAll was called with the non-empty list  
 *verify*(documentServiceRequestRepository).saveAll(documentServiceRequests);  
  
 *verify*(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
 }  
  
 //Method -1 Testcase:5  
 //5. Exception Handling:  
 //• Test scenarios where exceptions might be thrown, such as when there's an issue with the database or JSON processing.  
 //• Ensure that the getEventDataResponse method handles exceptions gracefully and returns the expected results or handles them appropriately  
 // (e.g., logging, error responses).  
 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Database Exception Handling:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 @Test  
 public void testDatabaseExceptionHandling() throws Exception {  
 // Mock behavior for dmEventRepository to throw a DataAccessException  
 *when*(dmEventRepository.save(*any*())).thenThrow(DataAccessException.class);  
  
 // Create sample EventRequest and EventResponse  
 EventRequest eventRequest = new EventRequest();  
 EventResponse eventResponse = new EventResponse();  
  
 // Call the method under test and expect it to handle the exception gracefully  
 eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Add assertions or verifications for how the exception is handled, e.g., logging or error response checks  
 }  
 //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_JSON Processing Exception Handling:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 @Test  
 public void testJsonProcessingExceptionHandling() throws Exception {  
 // Mock behavior for objectMapper to throw a JsonProcessingException  
 *when*(objectMapper.writeValueAsString(*any*())).thenThrow(JsonProcessingException.class);  
  
 // Create sample EventRequest and EventResponse  
 EventRequest eventRequest = new EventRequest();  
 EventResponse eventResponse = new EventResponse();  
  
 // Call the method under test and expect it to handle the exception gracefully  
 eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Add assertions or verifications for how the exception is handled, e.g., logging or error response checks  
 }  
 //Method-1 testcase:6  
 //10. Boundary Testing:  
 //Test the limits of the system, such as the maximum number of events or event statuses that can be processed in a single call.  
 @Test  
 public void testLargeNumberOfDocumentServiceRequests() throws Exception {  
 // Create a sample EventRequest and EventResponse  
 EventRequest eventRequest = new EventRequest();  
 EventResponse eventResponse = new EventResponse();  
  
 // Create a list of DocumentServiceRequest objects with a large number of items  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
 for (int i = 0; i < 1000; i++) {  
 documentServiceRequests.add(new DocumentServiceRequest());  
 }  
  
 // Mock behavior for eventMapper  
 *when*(eventMapper.eventRequestToDocumentEventEntity(eventRequest)).thenReturn(new DmEvent());  
 // when(eventMapper.updateNewEventResponse(eventRequest, eventResponse)).thenReturn(eventResponse);  
 *doNothing*().when(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
  
 // Mock behavior for dmEventRepository  
 *when*(dmEventRepository.save(*any*(DmEvent.class))).thenReturn(new DmEvent());  
  
 // Mock behavior for dmEventStatusRepository  
 *when*(dmEventStatusRepository.save(*any*(DmEventStatus.class))).thenReturn(new DmEventStatus());  
  
 // Mock behavior for documentServiceRequestRepository (large list)  
 *when*(documentServiceRequestRepository.saveAll(*anyList*())).thenReturn(documentServiceRequests);  
  
 // Call the method under test  
 eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify interactions with dependencies  
 *verify*(eventMapper).eventRequestToDocumentEventEntity(eventRequest);  
 *verify*(dmEventRepository).save(*any*(DmEvent.class));  
 *verify*(dmEventStatusRepository).save(*any*(DmEventStatus.class));  
  
 // Verify that documentServiceRequestRepository.saveAll was called with the large list  
 *verify*(documentServiceRequestRepository).saveAll(documentServiceRequests);  
  
 *verify*(eventMapper).updateNewEventResponse(eventRequest, eventResponse);  
 }  
 //Method-1 testcase :7  
 //7. Edge Cases:  
 //• Test edge cases such as providing null EventRequest or EventResponse objects.  
 //• Test the case where some of the dependencies (eventMapper, repositories) return null values.  
 //• Verify that the code handles these cases gracefully without causing errors.  
 @Test  
 public void testNullEventRequestAndEventResponse() throws Exception {  
 // Create sample null EventRequest and EventResponse  
 EventRequest eventRequest = null;  
 EventResponse eventResponse = null;  
  
 // Call the method under test with null inputs  
 Optional<EventResponse> result = eventStoreHelper.getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify that the result is an empty Optional  
 *assertEquals*(Optional.*empty*(), result);  
  
 // Ensure that no interactions with dependencies occurred  
 *verifyNoInteractions*(eventMapper, dmEventRepository, dmEventStatusRepository, documentServiceRequestRepository);  
 }  
 //Method-1 Testcase:8:2:1 for private persistDocSerReqAndStatusRecords()  
 //  
 @Test  
 public void testPersistDocSerReqAndStatusRecordsWithValidInput() {  
 // Test when the list of DocumentServiceRequest objects is not empty and saves successfully  
 List<DocumentServiceRequest> requestList = createSampleDocumentServiceRequests();  
  
 *when*(documentServiceRequestRepository.saveAll(requestList)).thenReturn(requestList);  
  
 List<DocumentServiceRequestStatus> statusList = new ArrayList<>();  
 *when*(eventMapper.mapToDocumentServiceRequestStatusEntity(requestList)).thenReturn(statusList);  
 *when*(documentServiceRequestStatusRepository.saveAll(statusList)).thenReturn(statusList);  
  
 eventStoreHelper.persistDocSerReqAndStatusRecords(requestList);  
  
 // Verify that the repositories' saveAll methods were called  
 *verify*(documentServiceRequestRepository, *times*(1)).saveAll(requestList);  
 *verify*(documentServiceRequestStatusRepository, *times*(1)).saveAll(statusList);  
 }  
 //Method -1 test case -9:2:2  
 @Test  
 public void testPersistDocSerReqAndStatusRecordsWithEmptyInput() {  
 // Test when the list of DocumentServiceRequest objects is empty  
 List<DocumentServiceRequest> requestList = Collections.*emptyList*();  
  
 eventStoreHelper.persistDocSerReqAndStatusRecords(requestList);  
  
 // Verify that the repositories' saveAll methods were not called since the input list is empty  
 *verify*(documentServiceRequestRepository, *never*()).saveAll(*anyList*());  
 *verify*(documentServiceRequestStatusRepository, *never*()).saveAll(*anyList*());  
 }  
 //Method -1 Testcase :10:3:1 fortestGetPrintFileRequest()  
 //1. Valid Input with Documents:  
 //• Provide an EventRequest object with a valid EventPayload containing documents.  
 //• Ensure that the method correctly parses and maps the documents into DocumentServiceRequest objects.  
 //• Verify that the returned list of DocumentServiceRequest objects is as expected.  
 @Test  
 public void testGetPrintFileRequestWithValidInput() throws Exception {  
 // Sample EventRequest with valid EventPayload containing documents  
 EventRequest eventRequest = createSampleEventRequestWithDocuments();  
  
 // Mock the behavior of objectMapper.readTree to return a valid ArrayNode  
 ArrayNode arrayNode = new ObjectMapper().createArrayNode();  
 *when*(objectMapper.readTree(*anyString*())).thenReturn(arrayNode);  
  
 // Mock the behavior of eventMapper to return a list of DocumentServiceRequest  
 List<DocumentServiceRequest> expectedDocumentServiceRequests = createSampleDocumentServiceRequests();  
 *when*(eventMapper.mapToDocumentServiceRequestEntity(*anyList*())).thenReturn(expectedDocumentServiceRequests);  
  
 // Call the method under test  
 List<DocumentServiceRequest> documentServiceRequests = eventStoreHelper.getPrintFileRequest(eventRequest);  
  
 // Verify that objectMapper.readTree and eventMapper were called with the expected arguments  
 *verify*(objectMapper).readTree(*anyString*());  
 *verify*(eventMapper).mapToDocumentServiceRequestEntity(*anyList*());  
  
 // Verify the result  
 *assertEquals*(expectedDocumentServiceRequests, documentServiceRequests);  
 }  
  
 // Helper methods to create sample data  
  
 private EventRequest createSampleEventRequestWithDocuments() {  
 EventRequest eventRequest = new EventRequest();  
 // var eventPayload = new EventPayload();  
 ObjectNode data = new ObjectMapper().createObjectNode();  
 ArrayNode documents = new ObjectMapper().createArrayNode();  
  
 ObjectNode document1 = new ObjectMapper().createObjectNode();  
 document1.put("name", "Document1");  
 document1.put("content", "Content1");  
 documents.add(document1);  
  
 ObjectNode document2 = new ObjectMapper().createObjectNode();  
 document2.put("name", "Document2");  
 document2.put("content", "Content2");  
 documents.add(document2);  
  
 data.set("documents", documents);  
 // eventPayload.setData(data);  
 // eventRequest.setEventDataRequest(eventPayload);  
  
 return eventRequest;  
 }  
 private List<DocumentServiceRequest> createSampleDocumentServiceRequests() {  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
  
 // Sample data from createSampleEventRequestWithDocuments  
 String[] documentNames = {"Document1", "Document2"};  
 String[] documentContents = {"Content1", "Content2"};  
  
 for (int i = 0; i < documentNames.length; i++) {  
 DocumentServiceRequest documentServiceRequest = new DocumentServiceRequest();  
 documentServiceRequest.setName(documentNames[i]);  
 documentServiceRequest.setContent(documentContents[i]);  
 documentServiceRequests.add(documentServiceRequest);  
 }  
  
 return documentServiceRequests;  
 }  
 //Method -1 testcase-11.3:2  
 //2. Valid Input without Documents:  
 //• Provide an EventRequest object with a valid EventPayload but without any documents.  
 //• Ensure that the method handles this scenario gracefully and returns an empty list or null.  
 @Test  
 public void testGetPrintFileRequestWithValidInputWithoutDocuments() throws Exception {  
 // Sample EventRequest with a valid EventPayload but without documents  
 EventRequest eventRequest = createSampleEventRequestWithoutDocuments();  
  
 // Call the method under test  
 List<DocumentServiceRequest> documentServiceRequests = eventStoreHelper.getPrintFileRequest(eventRequest);  
  
 // Verify that objectMapper.readTree and eventMapper were not called  
 *verify*(objectMapper, *never*()).readTree(*anyString*());  
 *verify*(eventMapper, *never*()).mapToDocumentServiceRequestEntity(*anyList*());  
  
 // Verify that the result is null or an empty list  
 *assertNull*(documentServiceRequests);  
 }  
  
 // Helper method to create a sample EventRequest without documents  
  
 private EventRequest createSampleEventRequestWithoutDocuments() {  
 EventRequest eventRequest = new EventRequest();  
 // EventPayload eventPayload = new EventPayload();  
 ObjectNode data = new ObjectMapper().createObjectNode();  
  
 // No documents in the EventPayload  
 //eventPayload.setData(data);  
 //eventRequest.setEventDataRequest(eventPayload);  
  
 return eventRequest;  
 }  
 //Method-1 Testcase:13:3:3  
 //3. Null EventRequest:  
 //• Pass a null EventRequest to the method.  
 //• Verify that the method handles this case and returns an appropriate result (e.g., empty list or null).  
 @Test  
 public void testGetPrintFileRequestWithNullEventRequest() throws Exception {  
 // Pass a null EventRequest to the method  
 EventRequest eventRequest = null;  
  
 // Call the method under test  
 List<DocumentServiceRequest> documentServiceRequests = eventStoreHelper.getPrintFileRequest(eventRequest);  
  
 // Verify that objectMapper.readTree and eventMapper were not called  
 *verify*(objectMapper, *never*()).readTree(*anyString*());  
 *verify*(eventMapper, *never*()).mapToDocumentServiceRequestEntity(*anyList*());  
  
 // Verify that the result is null  
 *assertNull*(documentServiceRequests);  
 }  
 //Method-1 Testcase :14:3:4  
 //4. Null EventPayload:  
 //• Provide an EventRequest object with a null EventPayload.  
 //• Ensure that the method handles this case and returns an appropriate result.  
 @Test  
 public void testGetPrintFileRequestWithNullEventPayload() throws Exception {  
 // Create an EventRequest with a null EventPayload  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setEventDataRequest(new EventDataRequest());  
  
 // Call the method under test  
 List<DocumentServiceRequest> documentServiceRequests = eventStoreHelper.getPrintFileRequest(eventRequest);  
  
 // Verify that objectMapper.readTree and eventMapper were not called  
 *verify*(objectMapper, *never*()).readTree(*anyString*());  
 *verify*(eventMapper, *never*()).mapToDocumentServiceRequestEntity(*anyList*());  
  
 // Verify that the result is null or an empty list, depending on your desired behavior  
 *assertNull*(documentServiceRequests);  
 }  
 //Method-1 Testcase:15:4:1  
 //testGetEventStatusForPrintFileEventKeyWithDocumentRequests  
 @Test  
 public void testGetEventStatusForPrintFileEventKeyWithDocumentRequests() {  
 // Test when eventKey is for a Print File event and documentServiceRequests is not empty  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
 documentServiceRequests.add(new DocumentServiceRequest());  
 String eventStatus = eventStoreHelper.getEventStatus(documentServiceRequests, DocumentGeneratorEventStoreConstants.*EVENT\_KEY\_PRINT\_FILE*);  
 *assertEquals*(EventStatusType.*DOCUMENT\_REQUEST\_RECEIVED*.name(), eventStatus);  
 }  
 //Method-1 Testcase:16:4:2  
 //testGetEventStatusForNonPrintFileEventKeyWithDocumentRequests  
 @Test  
 public void testGetEventStatusForNonPrintFileEventKeyWithDocumentRequests() {  
 // Test when eventKey is not for a Print File event and documentServiceRequests is not empty  
 List<DocumentServiceRequest> documentServiceRequests = new ArrayList<>();  
 documentServiceRequests.add(new DocumentServiceRequest());  
 String eventStatus = eventStoreHelper.getEventStatus(documentServiceRequests, "SomeOtherEventKey");  
 *assertEquals*(EventStatusType.*EVENT\_RECEIVED*.name(), eventStatus);  
 }  
 //Method-1 Testcase:17:4:3  
 //testGetEventStatusForNullDocumentRequests  
 @Test  
 public void testGetEventStatusForNullDocumentRequests() {  
 // Test when documentServiceRequests is null  
 String eventStatus = eventStoreHelper.getEventStatus(null, "SomeEventKey");  
 *assertEquals*(EventStatusType.*EVENT\_RECEIVED*.name(), eventStatus);  
 }  
 //Method-1 Testcase:18:4:4  
 //testGetEventStatusForEmptyDocumentRequests  
 @Test  
 public void testGetEventStatusForEmptyDocumentRequests() {  
 // Test when documentServiceRequests is an empty list  
 String eventStatus = eventStoreHelper.getEventStatus(new ArrayList<>(), "SomeEventKey");  
 *assertEquals*(EventStatusType.*EVENT\_RECEIVED*.name(), eventStatus);  
 }  
 //Method-1 Testcase:19:5:1  
 //testArrayNodeToDtoWithValidInput  
 @Test  
 public void testArrayNodeToDtoWithValidInput() throws JsonProcessingException {  
 // Test when arrayNode contains valid JSON nodes  
 ArrayNode arrayNode = *mock*(ArrayNode.class);  
 JsonNode jsonNode1 = *mock*(JsonNode.class);  
 JsonNode jsonNode2 = *mock*(JsonNode.class);  
 *when*(arrayNode.iterator()).thenReturn(Arrays.*asList*(jsonNode1, jsonNode2).iterator());  
  
 PrintDocumentsServiceRequest request1 = new PrintDocumentsServiceRequest();  
 PrintDocumentsServiceRequest request2 = new PrintDocumentsServiceRequest();  
 *when*(objectMapper.treeToValue(jsonNode1, PrintDocumentsServiceRequest.class)).thenReturn(request1);  
 *when*(objectMapper.treeToValue(jsonNode2, PrintDocumentsServiceRequest.class)).thenReturn(request2);  
  
 List<PrintDocumentsServiceRequest> result = eventStoreHelper.arrayNodeToDto(arrayNode, objectMapper);  
  
 *assertEquals*(2, result.size());  
 *assertEquals*(request1, result.get(0));  
 *assertEquals*(request2, result.get(1));  
 }  
 //Method-1 Testcase:20:5:2  
 //testArrayNodeToDtoWithEmptyArray  
 @Test  
 public void testArrayNodeToDtoWithEmptyArray() throws JsonProcessingException {  
 // Test when arrayNode is empty  
 ArrayNode arrayNode = *mock*(ArrayNode.class);  
 *when*(arrayNode.iterator()).thenReturn(Collections.*emptyIterator*());  
  
 List<PrintDocumentsServiceRequest> result = eventStoreHelper.arrayNodeToDto(arrayNode, objectMapper);  
  
 *assertEquals*(0, result.size());  
 }  
  
 }