**Controller Class :** Method 5 (line no 141)

* **Annotations**:

This code uses annotations to define various aspects of the API endpoint:

* **@Operation**: Provides a summary description for the API operation. In this case, it's a description of what this API endpoint does.
* **@ApiResponses**: Specifies the possible responses for this API operation. In this case, it defines a single response with a 200 HTTP status code and a description indicating that "notify" details were retrieved successfully. It also specifies that the response will be in JSON format and will follow the schema defined by the EventResponse class.
* **@GetMapping**: Indicates that this method should handle HTTP GET requests. The URL path for this endpoint is constructed by concatenating several constants defined in the DocumentGeneratorEventStoreConstants class, which likely represent parts of the URL path.
* **Method Signature**: This method is defined to take two parameters:

**HttpServletRequest httpRequest**: An object representing the HTTP request. This is typically used to access information about the incoming request.

**@NotNull @PathVariable final String eventId**: A path variable named **eventId** that is marked as not nullable (**@NotNull**). This means the **eventId** must be present in the URL path, and it cannot be null.

**Logging**: This line logs a message using a logger (presumably from a logging framework like Log4j or SLF4J). It logs the message "**Getting Notify data from eventId:** " followed by the sanitized value of the **eventId**. The purpose of this log message is likely for debugging and tracking API usage.

**Service Method Call**: This line invokes a method named **fetchNotifyDetailsByEventId** on an instance of **documentGeneratorEventStoreService**. This method is expected to return an **Optional<EventResponse>**. It appears to be retrieving "**notify**" details based on the provided **eventId**.

**Response Mapping**: Finally, this line returns the result of calling the **eventResponseMapper** method, passing in the **httpRequest** and **eventDataResponse** as parameters. This method is responsible for mapping the retrieved data into an appropriate response format and returning it as an instance of **ResponseEntity<EventResponse>**.

**Test Case**: 2 test cases for Method 5 :

**1. Success() :** Here’s what the code is doing.

**-retrieveNotifyResponseByEventId\_Success**: This is a method or function name. The name suggests that this method is responsible for testing a successful scenario of the **retrieveNotifyResponseByEventId** operation. The **\_Success** part indicates that the test case is focused on the expected successful result.

* **Arrange**: In this section, the test is setting up the initial conditions and preparing the data for the test case.

**eventId**: A sample event ID.

**sanitizedEventId**: The expected sanitized version of the event ID.

**Mocking**: This part of the test is using Mockito to mock the behavior of the genericUtil object. It specifies that when the **sanitizeValues** method is called with the eventId, it should return the **sanitizedEventId**. This is done to isolate the controller's behavior from the actual **genericUti**l implementation.

**More Mocking**: Similarly, this part of the test is mocking the behavior of the **documentGeneratorEventStoreService**. It specifies that when the **fetchNotifyDetailsByEventId** method is called with the **sanitizedEventId**, it should return an **Optional** containing an **expectedResponse**. Again, this is done to isolate the controller's behavior from the actual service implementation.

* **Act**: This section is executing the method being tested. It calls the **retrieveNotifyResponseByEventId** method of the controller with the provided **httpRequest** and **eventId** parameters.
* **Assertions**: In this part, the test is making assertions to verify that the actual behavior matches the expected behavior:

**assertEquals(HttpStatus.OK, responseEntity.getStatusCode())**: This verifies that the HTTP status code of the responseEntity is equal to **HttpStatus.OK**, indicating a successful response.

**assertEquals(expectedResponse, responseEntity.getBody())**: This verifies that the body of the **responseEntity** is equal to the **expectedResponse**.

* **Verification**: Finally, this part uses Mockito's verify method to ensure that specific methods were called during the test:

**verify(genericUtil, times(1)).sanitizeValues(eventId)**: It verifies that the **sanitizeValues** method of **genericUtil** was called exactly once with the **eventId**.

**verify(documentGeneratorEventStoreService, times(1)).fetchNotifyDetailsByEventId(sanitizedEventId**): It verifies that the **fetchNotifyDetailsByEventId** method of **documentGeneratorEventStoreService** was called exactly once with the **sanitizedEventId**

**Test Case**: 2 test cases for Method 5 :

**2. Failure() :** Here’s what the code is doing.

**-retrieveNotifyResponseByEventId\_Failure()**: This is a method name specifically designed for testing a particular scenario in the codebase. It suggests that this test method is focused on verifying the failure scenario of the **retrieveNotifyResponseByEventId** operation..

* **Arrange**: In this section, the test is setting up the initial conditions and preparing the data for the test case.

**eventId**: A sample event ID.

**sanitizedEventId**: The expected sanitized version of the event ID.

**Mockito Initialization**: This line initializes **Mockito** for the current test class. Mockito is a popular framework for mocking objects and behaviors in unit tests. In this case, it prepares the test class to use **Mockito** to mock certain behavior.

**Mocking**: This part of the test is using Mockito to mock the behavior of the genericUtil object. It specifies that when the **sanitizeValues** method is called with the eventId, it should return the **sanitizedEventId**. This is done to isolate the controller's behavior from the actual **genericUti**l implementation.

**More Mocking**: Similarly, this part of the test is mocking the behavior of the **documentGeneratorEventStoreService**. It specifies that when the **fetchNotifyDetailsByEventId** method is called with the **sanitizedEventId**, it should return an empty **Optional**. This simulates a failure scenario where no data is found for the given event ID.

* **Act**: This section is executing the method being tested. It calls the **retrieveNotifyResponseByEventId** method of the controller with the provided **httpRequest** and **eventId** parameters.
* **Assertions**: In this part, the test is making assertions to verify that the actual behavior matches the expected behavior:

**assertEquals(HttpStatus.NOT\_FOUND, responseEntity.getStatusCode()):** This verifies that the HTTP status code of the responseEntity is equal to **HttpStatus.NOT\_FOUND**, indicating a failure scenario.

**assertNull(responseEntity.getBody()**): This ensures that the body of the **responseEntity** is null since no data was found in the failure scenario.

* **Verification**: Finally, this part uses Mockito's verify method to ensure that specific methods were called during the test:

**verify(genericUtil, times(1)).sanitizeValues(eventId)**: It verifies that the **sanitizeValues** method of **genericUtil** was called exactly once with the **eventId**.

**verify(documentGeneratorEventStoreService, times(1)).fetchNotifyDetailsByEventId(sanitizedEventId**): It verifies that the **fetchNotifyDetailsByEventId** method of **ocumentGeneratorEventStoreService** was called exactly once with the **sanitizedEventId**

// Method 5 line no 141 : 2 Test cases  
// 1. Success() API to retrieve notify details by eventId  
@Test  
public void retrieveNotifyResponseByEventId\_Success() {  
 // Arrange  
 String eventId = "sampleEventId";  
 String sanitizedEventId = "sanitizedEventId";  
  
 // Mock the behavior of GenericUtil  
 when(genericUtil.sanitizeValues(eventId)).thenReturn(sanitizedEventId);  
  
 // Mock the behavior of the service  
 EventResponse expectedResponse = new EventResponse();  
 when(documentGeneratorEventStoreService.fetchNotifyDetailsByEventId(sanitizedEventId))  
 .thenReturn(Optional.of(expectedResponse));  
  
 // Act  
 ResponseEntity<EventResponse> responseEntity = controller.retrieveNotifyResponseByEventId(httpRequest, eventId);  
  
 // Assert  
 assertEquals(HttpStatus.OK, responseEntity.getStatusCode());  
 assertEquals(expectedResponse, responseEntity.getBody());  
  
 // Verify that GenericUtil.sanitizeValues was called with eventId  
 verify(genericUtil, times(1)).sanitizeValues(eventId);  
  
 // Verify that documentGeneratorEventStoreService.fetchNotifyDetailsByEventId was called with sanitizedEventId  
 verify(documentGeneratorEventStoreService, times(1)).fetchNotifyDetailsByEventId(sanitizedEventId);  
}

// Method 5 line no 141 : 2 Test cases  
// 2. Failure() API to retrieve notify details by eventId  
@Test  
public void retrieveNotifyResponseByEventId\_Failure() {  
 // Arrange  
 String eventId = "sampleEventId";  
 String sanitizedEventId = "sanitizedEventId";  
  
 // Initialize Mockito  
 MockitoAnnotations.initMocks(this);  
  
 // Mock the behavior of GenericUtil  
 when(genericUtil.sanitizeValues(eventId)).thenReturn(sanitizedEventId);  
  
 // Mock the behavior of the service to return an empty Optional (failure scenario)  
 when(documentGeneratorEventStoreService.fetchNotifyDetailsByEventId(sanitizedEventId))  
 .thenReturn(Optional.empty());  
  
 // Act  
 ResponseEntity<EventResponse> responseEntity = controller.retrieveNotifyResponseByEventId(httpRequest, eventId);  
  
 // Assert  
 assertEquals(HttpStatus.NOT\_FOUND, responseEntity.getStatusCode());  
 // Ensure the response body is null since no data was found  
 assertNull(responseEntity.getBody());  
  
 // Verify that GenericUtil.sanitizeValues was called with eventId  
 verify(genericUtil, times(1)).sanitizeValues(eventId);  
  
 // Verify that documentGeneratorEventStoreService.fetchNotifyDetailsByEventId was called with sanitizedEventId  
 verify(documentGeneratorEventStoreService, times(1)).fetchNotifyDetailsByEventId(sanitizedEventId);  
}