

```
1 package com.inty.eventregistration.api;
2
3 @import java.util.List;[]
4
5
6 @RestController
7 @Validated
8 @RequestMapping(value="/event-api")
9 public class EventAPI {
10
11     @Autowired
12     private EventService eventService;
13
14     @Autowired
15     private Environment environment;
16
17     // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
18     @PostMapping(value="/events")
19     public ResponseEntity<String> registerParticipant(
20         @Valid @RequestBody ParticipantDTO participantDTO) throws EventRegistrationException {
21         // your code goes here
22         return null;
23         Integer id = eventService.registerParticipant(participantDTO);
24         return new ResponseEntity<>(environment.getProperty("API.REGISTRATION_SUCCESS")+id,HttpStatus.CREATED);
25     }
26
27     // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
28     @GetMapping(value= "/events/{venue}")
29     public ResponseEntity<List<ParticipantDTO>> getParticipantsByEventVenue(@PathVariable
30         @Pattern(regexp="[A-Z][0-9](-Hall)",message = "{event.venue.invalid}") String venue)
31         throws EventRegistrationException {
32         // your code goes here
33         return null;
34         List<ParticipantDTO> list = eventService.getParticipantsByEventVenue(venue);
35         return new ResponseEntity<List<ParticipantDTO>>(list,HttpStatus.OK);
36     }
37
38 }
```

File Edit Source Refactor Navigate Search Project Run Window Help

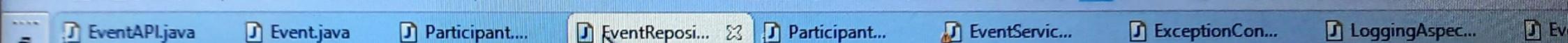


EventAPI.java Event.java X Participant... EventReposit... Participant... EventService... ExceptionCon... LoggingAspec... Ev

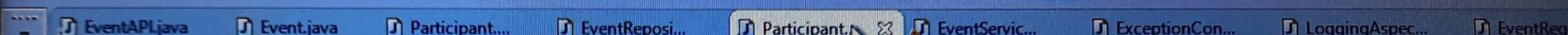
```
1 package com.infy.eventregistration.entity;
2
3 import java.time.LocalDate;
4
5 @Entity
6 @Table(name="event")
7 public class Event
8 {
9     @Id
10    private Integer eventId;
11    private String name;
12    private LocalDate eventDate;
13    private String venue;
14    private Integer maxCount;
```

The screenshot shows a Java IDE interface with multiple tabs at the top. The current tab is 'Participant.java', which is highlighted with a blue background. The code editor below displays the 'Participant' entity class. The code uses annotations from the 'javax.persistence' and 'java.time' packages. The 'ParticipantId' field is annotated with @Id and @GeneratedValue(strategy = GenerationType.IDENTITY). The 'event' field is annotated with @ManyToOne(cascade = CascadeType.ALL) and @JoinColumn(name = "event_id"). The class has methods for getting and setting the participant ID.

```
1 package com.infy.eventregistration.entity;
2
3 import java.time.LocalDate;
4
5
6 @Entity
7 @Table(name="participant")
8 public class Participant
9 {
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private Integer participantId;
13     private String name;
14     private String emailId;
15     private String gender;
16     private LocalDate registrationDate;
17
18     @ManyToOne(cascade = CascadeType.ALL)
19     @JoinColumn(name="event_id")
20     private Event event;
21
22
23     public Integer getParticipantId()
24     {
25         return participantId;
26     }
27
28     public void setParticipantId(Integer participantId)
29     {
30         this.participantId = participantId;
31     }
32 }
```



```
1 package com.infy.eventregistration.repository;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 public interface EventRepository extends CrudRepository<Event, Integer>{
6     // your code goes here
7     public Event findByName(String name);
8 }
9
10 }
```



```
1 package com.infy.eventregistration.repository;
2
3 import java.util.List;
4
5 public interface ParticipantRepository extends CrudRepository<Participant, Integer>{
6     // your code goes here
7     // List<Participant> findByVenue(String venue);
8
9     @Query("SELECT p FROM Participant p WHERE p.event.venue =: venue ORDER BY p.registrationDate DESC")
10    public List<Participant> findByVenue(@Param("venue") String venue);
11
12    public List<Participant> findByEvent(Event event);
13 }
14
15
16
17
18
19
20
21 }
22
```

```
1 package com.infy.eventregistration.service;
2
3 import java.util.ArrayList;
4
5
6 @Service(value = "eventService")
7 @Transactional
8 public class EventServiceImpl implements EventService {
9
10    @Autowired
11    private EventRepository eventRepository;
12
13    @Autowired
14    private ParticipantRepository participantRepository;
15
16    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
17    @Override
18    public Integer registerParticipant(ParticipantDTO participantDTO) throws EventRegistrationException {
19        // your code goes here
20        // return null;
21        Event event =eventRepository.findByName(participantDTO.getEventDTO().getName());
22        if(event==null) {
23            throw new EventRegistrationException("Service.EVENT_UNAVAILABLE");
24        }
25        List<Participant> listP = participantRepository.findByEvent(event);
26
27        if(listP.size()>=event.getMaxCount()) {
28            throw new EventRegistrationException("Service.REGISTRATION_CLOSED");
29        }
30
31
32        if(participantDTO.getRegistrationDate().isBefore(event.getEventDate().minusDays(2))) {
33            throw new EventRegistrationException("Service.REGISTRATION_CLOSED");
34        }
35
36
37        Participant part = new Participant();
38        part.setEmailId(participantDTO.getEmailId());
39        part.setGender(participantDTO.getGender());
40
41
42
43
44
45
46
47
48
49
50
51
52
53
```

```
48
49
50
51     Participant part = new Participant();
52     part.setEmailId(participantDTO.getEmailId());
53     part.setGender(participantDTO.getGender());
54     part.setName(participantDTO.getName());
55     part.setParticipantId(participantDTO.getParticipantId());
56     part.setRegistrationDate(participantDTO.getRegistrationDate());
57
58     part.setEvent(event);
59
60     Participant part1 = participantRepository.save(part);
61     return part1.getParticipantId();
62 }
63
64 // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
65
66 @Override
67 public List<ParticipantDTO> getParticipantsByEventVenue(String venue) throws EventRegistrationException {
68     // your code goes here
69     List<Participant> opt1 = participantRepository.findByVenue(venue);
70
71     if(opt1.isEmpty()) {
72         throw new EventRegistrationException("Service.PARTICIPANTS_UNAVAILABLE");
73     }
74
75     List<ParticipantDTO>list = new LinkedList<>();
```

```
26  
27 @Autowired  
28 private ParticipantRepository participantRepository;  
29  
30 // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD  
31 @Override  
32 public Integer registerParticipant(ParticipantDTO participantDTO) throws EventRegistrationException {  
33     // your code goes here  
34     return null;  
35     Event event = eventRepository.findByName(participantDTO.getEventDTO().getName());  
36     if(event==null) {  
37         throw new EventRegistrationException("Service.EVENT_UNAVAILABLE");  
38     }  
39     List<Participant> listP = participantRepository.findByEvent(event);  
40  
41     if(listP.size()>=event.getMaxCount()) {  
42         throw new EventRegistrationException("Service.REGISTRATION_CLOSED");  
43     }  
44  
45  
46     if(participantDTO.getRegistrationDate().isBefore(event.getEventDate().minusDays(2))) {  
47         throw new EventRegistrationException("Service.REGISTRATION_CLOSED");  
48     }  
49  
50  
51     Participant part = new Participant();  
52     part.setEmailId(participantDTO.getEmailId());  
53     part.setGender(participantDTO.getGender());  
54     part.setName(participantDTO.getName());  
55     part.setParticipantId(participantDTO.getParticipantId());  
56     part.setRegistrationDate(participantDTO.getRegistrationDate());  
57  
58     part.setEvent(event);  
59  
60     Participant part1 = participantRepository.save(part);  
61     return part1.getParticipantId();  
62 }  
63  
64 // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
```

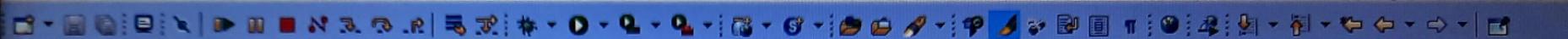
Line: 52

```
61     return participantId();
62 }
63
64 // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
65 @Override
66 public List<ParticipantDTO> getParticipantsByEventVenue(String venue) throws EventRegistrationException {
67     // your code goes here
68     //return null;
69     List<Participant> opt1 = participantRepository.findByVenue(venue);
70
71     if(opt1.isEmpty()) {
72         throw new EventRegistrationException("Service.PARTICIPANTS_UNAVAILABLE");
73     }
74
75     List<ParticipantDTO>list = new LinkedList<>();
76     for(Participant p:opt1) {
77
78         ParticipantDTO pDTO = new ParticipantDTO();
79         pDTO.setEmailId(p.getEmailId());
80         pDTO.setGender(p.getGender());
81         pDTO.setName(p.getName());
82         pDTO.setParticipantId(p.getParticipantId());
83         pDTO.setRegistrationDate(p.getRegistrationDate());
84
85         EventDTO event = new EventDTO();
86         event.setDate(p.getEvent().getEventDate());
87         event.setId(p.getEvent().getEventId());
88         event.setMaxCount(p.getEvent().getMaxCount());
89         event.setName(p.getEvent().getName());
90         event.setVenue(p.getEvent().getVenue());
91
92         pDTO.setEventDTO(event);
93         list.add(pDTO);
94     }
95     //Collections.sort(list);
96     //list.sort((p1,p2)->p2.getRegistrationDate().compareTo(p1.getRegistrationDate()));
97     return list;
98 }
99 }
```

```
1 package com.infy.eventregistration.utility;
2
3 import java.util.stream.Collectors;
4
5
6 @RestControllerAdvice
7 public class ExceptionControllerAdvice {
8
9     private static final Log LOGGER = LogFactory.getLog(ExceptionControllerAdvice.class);
10
11     @Autowired
12     private Environment environment;
13
14     // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
15     @ExceptionHandler(EventRegistrationException.class)
16     public ResponseEntity<ErrorInfo> eventRegistrationExceptionHandler(EventRegistrationException exception) {
17         LOGGER.error(exception.getMessage(), exception);
18         ErrorInfo errorInfo = new ErrorInfo();
19         errorInfo.setErrorCode(HttpStatus.BAD_REQUEST.value());
20         errorInfo.setErrorMessage(environment.getProperty(exception.getMessage()));
21         return new ResponseEntity<>(errorInfo, HttpStatus.BAD_REQUEST);
22     }
23
24     // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
25     @ExceptionHandler(Exception.class)
26     public ResponseEntity<ErrorInfo> generalExceptionHandler(Exception exception) {
27         LOGGER.error(exception.getMessage(), exception);
28         ErrorInfo errorInfo = new ErrorInfo();
29         errorInfo.setErrorCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
30         errorInfo.setErrorMessage(environment.getProperty("General.EXCEPTION_MESSAGE"));
31         return new ResponseEntity<>(errorInfo, HttpStatus.INTERNAL_SERVER_ERROR);
32     }
33
34     // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
35     @ExceptionHandler({MethodArgumentNotValidException.class, ConstraintViolationException.class})
36     public ResponseEntity<ErrorInfo> validatorExceptionHandler(Exception exception) {
37         LOGGER.error(exception.getMessage(), exception);
38         String errorMsg;
39         if (exception instanceof MethodArgumentNotValidException) {
40             MethodArgumentNotValidException mainException = (MethodArgumentNotValidException) exception;
41             errorMsg = mainException.getBindingResult().getFieldErrors().stream()
42                 .map(validatorError -> validatorError.getField() + " : " + validatorError.getDefaultMessage())
43                 .collect(Collectors.toList());
44             errorInfo.setErrorMessage(errorMsg);
45         } else {
46             errorMsg = exception.getMessage();
47             errorInfo.setErrorMessage(errorMsg);
48         }
49         return new ResponseEntity<>(errorInfo, HttpStatus.INTERNAL_SERVER_ERROR);
50     }
51 }
```

```
1 package com.infy.eventregistration.utility;
2
3③ import org.apache.commons.logging.Log;④
4
5 @Component
6 @Aspect
7 public class LoggingAspect {
8
9
10    private static final Log LOGGER = LogFactory.getLog(LoggingAspect.class);
11
12
13    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
14
15    @AfterThrowing(pointcut = "execution(* com.infy.eventregistration.service.*Impl.*(..))"
16                  ,throwing = "exception")
17    public void logServiceException(EventRegistrationException exception) {
18        // your code goes here
19        LOGGER.error(exception.getMessage(),exception);
20    }
21
22
23    }
24
25 }
```

```
23  
24  
25  
26 @SpringBootTest  
27 public class EventRegistrationApplicationTests {  
28  
29  
30    @Mock  
31    private EventRepository eventRepository;  
32    @Mock  
33    private ParticipantRepository participantRepository;  
34    @InjectMocks  
35    private EventService eventService = new EventServiceImpl();  
36  
37    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD  
38    @Test  
39    public void getParticipantsByEventVenueNoParticipantsFoundTest() {  
40        // your code goes here  
41        List<Participant> lsit = new ArrayList<>();  
42        String venue = "A1-HALL";  
43  
44        Mockito.when(participantRepository.findByVenue(Mockito.anyString())).thenReturn(lsit);  
45        EventRegistrationException exp = Assertions.assertThrows(EventRegistrationException.class, ()->eventService.getParticipantsByEventVenue(venue));  
46        Assertions.assertEquals("Service.PARTICIPANTS_UNAVAILABLE", exp.getMessage());  
47    }  
48  
49    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD  
50    @Test  
51    public void registerParticipantNoEventFoundTest() {  
52        // your code goes here  
53        Event e =new Event();  
54        e.setEventDate(LocalDate.now());  
55        e.setEventId(1234);  
56        e.setMaxCount(20);  
57        e.setName("Bello");  
58        e.setVenue("B1-Hall");  
59  
60        ParticipantDTO pdto = new ParticipantDTO();  
61    }
```



```
44     MOCKITO.when(participantRepository.findByVenue(Mockito.anyString())).thenReturn(list);
45     EventRegistrationException exp = Assertions.assertThrows(EventRegistrationException.class, ()->eventService.getParticipantsByEventVenue(venue));
46     Assertions.assertEquals("Service.PARTICIPANTS_UNAVAILABLE", exp.getMessage());
47 }
48
49 // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
50 @Test
51 public void registerParticipantNoEventFoundTest() {
52     // your code goes here
53     Event e =new Event();
54     e.setEventDate(LocalDate.now());
55     e.setEventId(1234);
56     e.setMaxCount(20);
57     e.setName("Bello");
58     e.setVenue("B1-Hall");
59
60     ParticipantDTO pdto = new ParticipantDTO();
61
62     pdto.setEmailId("gautam@infy.com");
63     pdto.setGender("Male");
64     pdto.setName("Gautam");
65     pdto.setRegistrationDate(LocalDate.now());
66
67     EventDTO edto = new EventDTO();
68     edto.setEventDate(e.getEventDate());
69     edto.setEventId(e.getEventId());
70     edto.setMaxCount(e.getMaxCount());
71     edto.setName(e.getName());
72     edto.setVenue(e.getVenue());
73
74     pdto.setEventDTO(edto);
75
76     Mockito.when(eventRepository.findByName(Mockito.anyString())).thenReturn(null);
77     EventRegistrationException exp = Assertions.assertThrows(EventRegistrationException.class, ()->eventService.registerParticipant(pdto));
78     Assertions.assertEquals("Service.EVENT_UNAVAILABLE", exp.getMessage());
79
80 }
81 }
```

The screenshot shows the Eclipse IDE interface with the title bar "Fa_mock - EventRegistration_ToTrainee/src/main/java/com/infy/eventregistration/dto/EventDTO.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Select. The perspective switcher shows "Java" is selected. The package explorer shows a folder icon. The Java editor tab bar includes EventDTO.java (selected), Event.java, Participant..., EventReposit..., Participant..., EventService..., ExceptionCon..., LoggingAspec..., and EventRegistration_ToTrainee. The code editor displays the following Java code:

```
1 package com.infy.eventregistration.dto;
2
3 import java.time.LocalDate;
4
5 public class EventDTO
6 {
7     private Integer eventId;
8
9     @Pattern(regexp="[A-Z][0-9](-Hall)+",message = "{event.venue.invalid}")
10    private String name;
11    private LocalDate eventDate;
12    private String venue;
13    private Integer maxCount;
14
15    public Integer getEventId()
16    {
17        return eventId;
18    }
19
20    public void setEventId(Integer eventId)
21    {
22        this.eventId = eventId;
23    }
24
25    public String getName()
26    {
27        return name;
28    }
29
30    }
31
```

```
EventDTO.java Participant... Participant... EventReposit... Participant... EventServic... ExceptionCon... LoggingAspec...
1 package com.infy.eventregistration.dto;
2
3 import java.time.LocalDate;
4
5 public class ParticipantDTO
6 {
7
8
9     private Integer participantId;
10
11
12     @NotNull(message= "{participant.name.notpresent}")
13     @Pattern(regexp= "[A-Z][a-z]*",message= "{participant.name.invalid}")
14     private String name;
15
16     @NotNull(message = "{participant.emailid.notpresent}")
17     @Pattern(regexp= "[A-Za-z0-9]+(@infy.com)",message = "{participant.emailid.invalid}")
18     private String emailId;
19
20     @NotNull(message= "{participant.gender.notpresent}")
21     @Pattern(regexp= "Female|Male|Others",message= "{participant.gender.invalid}")
22     private String gender;
23
24     @NotNull(message = "{participant.registrationdate.notpresent}")
25     @FutureOrPresent(message= "{participant.registrationdate.invalid}")
26     private LocalDate registrationDate;
27
28     @NotNull(message = "{participant.event.notpresent}")
29     private EventDTO eventDTO;
30
31     public Integer getParticipantId()
32     {
33         return participantId;
34     }
35
36     public void setParticipantId(Integer participantId)
37     {
38         this.participantId = participantId;
39     }
40
41 }
```