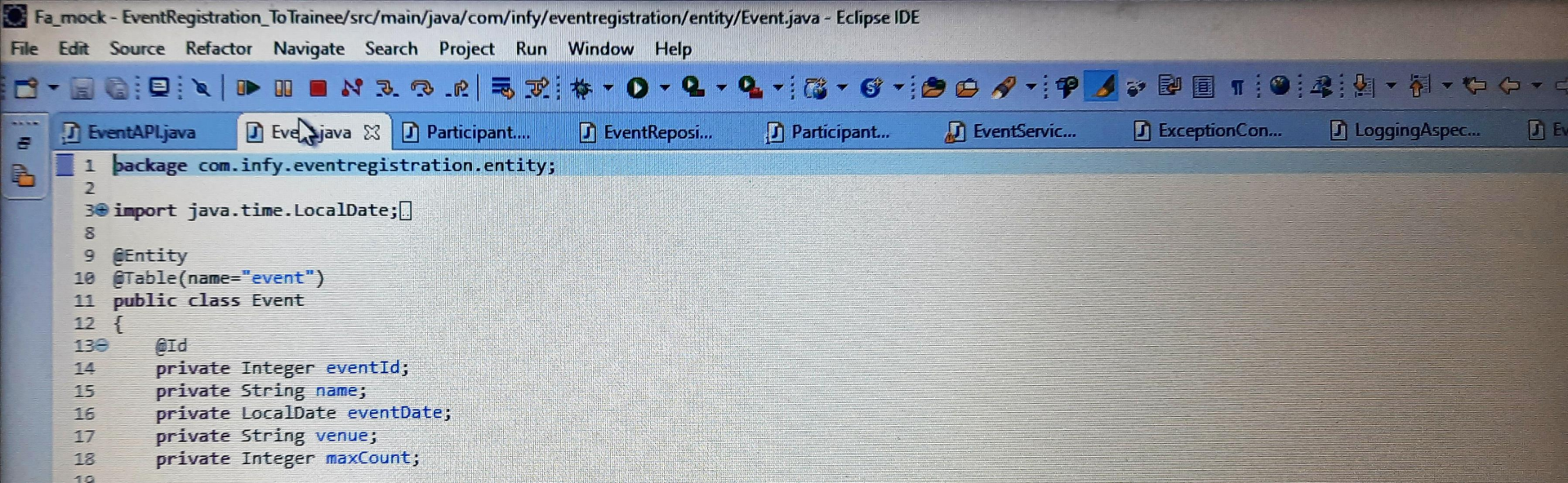
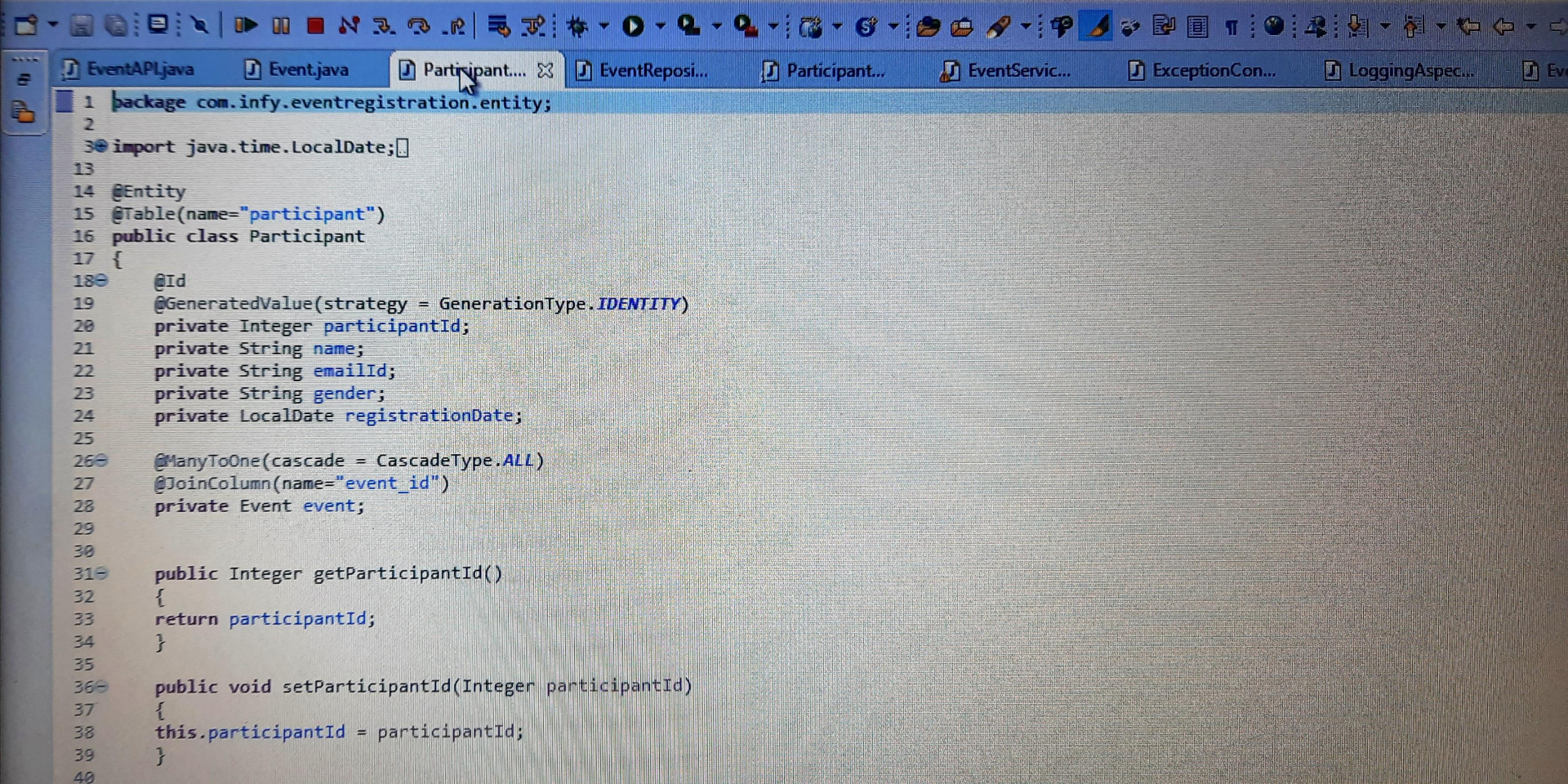
```java
package com.infy.eventregistration.api;

import java.util.List;


@RestController
@Validated
@RequestMapping(value="/event-api")
public class EventAPI {

    @Autowired
    private EventService eventService;

    @Autowired
    private Environment environment;

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @PostMapping(value="/events")
    public ResponseEntity<String> registerParticipant(
            @Valid  @RequestBody ParticipantDTO participantDTO) throws EventRegistrationException {
        // your code goes here
//      return null;
        Integer id = eventService.registerParticipant(participantDTO);
        return new ResponseEntity<>(environment.getProperty("API.REGISTRATION_SUCCESS")+id,HttpStatus.CREATED);
    }

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @GetMapping(value= "/events/{venue}")
    public ResponseEntity<List<ParticipantDTO>> getParticipantsByEventVenue(@PathVariable
            @Pattern(regexp="[A-Z][0-9](-Hall)",message = "{event.venue.invalid}") String venue)
            throws EventRegistrationException {
        // your code goes here
//      return null;
        List<ParticipantDTO> list = eventService.getParticipantsByEventVenue(venue);
        return new ResponseEntity<List<ParticipantDTO>>(list,HttpStatus.OK);
    }

}
```


Fa_mock - EventRegistration_ToTrainee/src/main/java/com/infy/eventregistration/entity/Event.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
EventAPI.java
Event.java
Participant...
EventReposi...
Participant...
EventServic...
ExceptionCon...
LoggingAspec...
1 package com.infy.eventregistration.entity;
2
3 import java.time.LocalDate;
8
9 @Entity
10 @Table(name="event")
11 public class Event
12 {
13 @Id
14 private Integer eventId;
15 private String name;
16 private LocalDate eventDate;
17 private String venue;
18 private Integer maxCount;


EventAPI.java
Event.java
Participant....
EventReposi...
Participant...
EventServic...
ExceptionCon...
LoggingAspec...
package com.infy.eventregistration.entity;
import java.time.LocalDate;
@Entity
@Table(name="participant")
public class Participant
{
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer participantId;
private String name;
private String emailId;
private String gender;
private LocalDate registrationDate;
@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name="event_id")
private Event event;
public Integer getParticipantId()
{
return participantId;
}
public void setParticipantId(Integer participantId)
{
this.participantId = participantId;
}

Fa_mock - EventRegistration_ToTrainee/src/main/java/com/infy/eventregistration/repository/EventRepository.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

EventAPI.java | Event.java | Participant.... | EventReposi... | Participant... | EventServic... | ExceptionCon... | LoggingAspec...

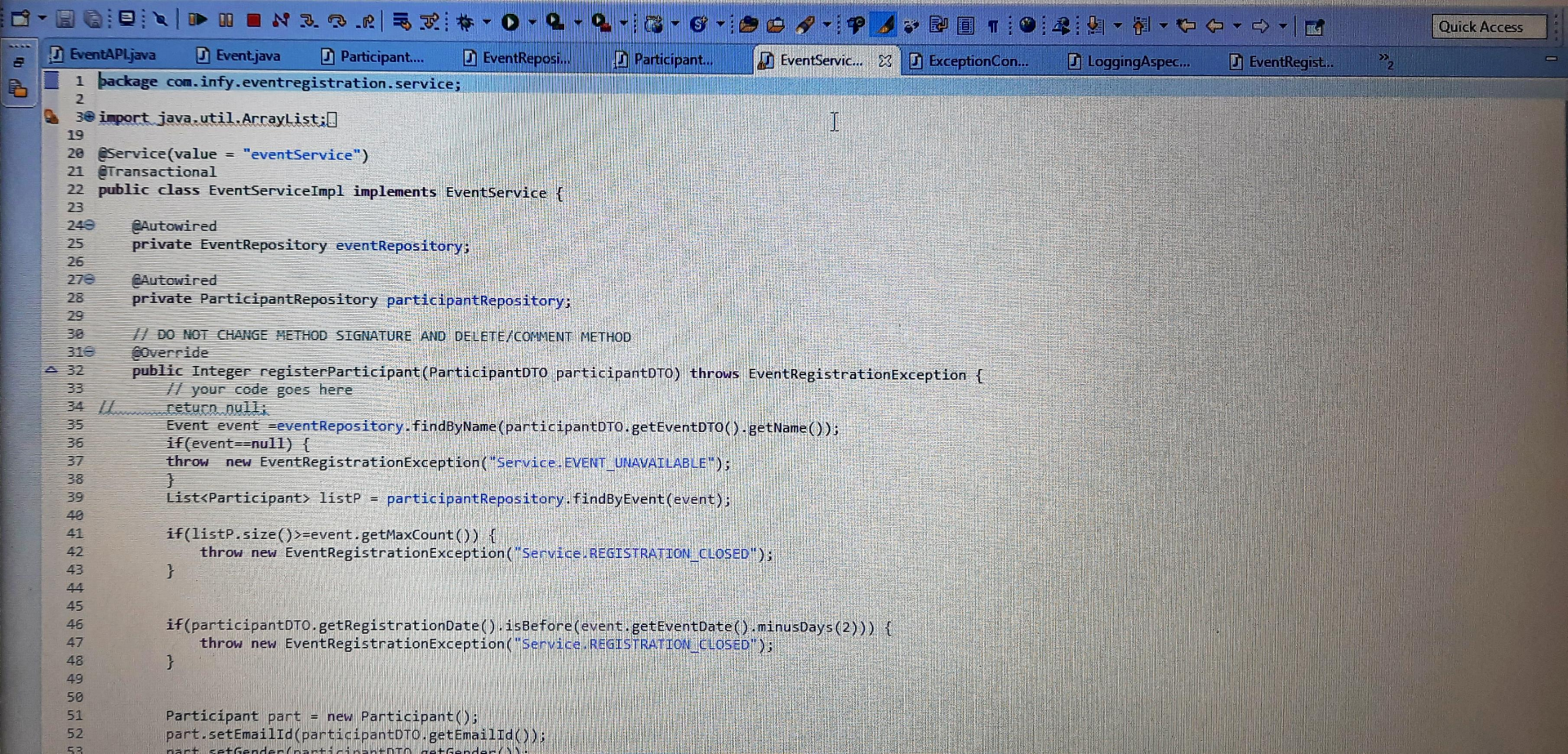
```
1  package com.infy.eventregistration.repository;
2
3  import org.springframework.data.repository.CrudRepository;
6
7  public interface EventRepository extends CrudRepository<Event, Integer>{
8      // your code goes here
9      public Event findByName(String name);
10 }
11
```

Fa_mock - EventRegistration_ToTrainee/src/main/java/com/infy/eventregistration/repository/ParticipantRepository.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

EventAPI.java | Event.java | Participant.... | EventReposi... | Participant... | EventServic... | ExceptionCon... | LoggingAspec... | EventReg

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


Quick Access
EventAPI.java
Event.java
Participant....
EventReposi...
Participant...
EventServic...
ExceptionCon...
LoggingAspec...
EventRegist...
1 package com.infy.eventregistration.service;
3 import java.util.ArrayList;
20 @Service(value = "eventService")
21 @Transactional
22 public class EventServiceImpl implements EventService {
24 @Autowired
25 private EventRepository eventRepository;
27 @Autowired
28 private ParticipantRepository participantRepository;
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);
41 if(listP.size()>=event.getMaxCount()) {
42 throw new EventRegistrationException("Service.REGISTRATION_CLOSED");
43 }
46 if(participantDTO.getRegistrationDate().isBefore(event.getEventDate().minusDays(2))) {
47 throw new EventRegistrationException("Service.REGISTRATION_CLOSED");
48 }
51 Participant part = new Participant();
52 part.setEmailId(participantDTO.getEmailId());

```
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
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<>();
```

```java
    @Autowired
    private ParticipantRepository participantRepository;

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @Override
    public Integer registerParticipant(ParticipantDTO participantDTO) throws EventRegistrationException {
        // your code goes here
//        return null;
        Event event =eventRepository.findByName(participantDTO.getEventDTO().getName());
        if(event==null) {
        throw  new EventRegistrationException("Service.EVENT_UNAVAILABLE");
        }
        List<Participant> listP = participantRepository.findByEvent(event);

        if(listP.size()>=event.getMaxCount()) {
            throw new EventRegistrationException("Service.REGISTRATION_CLOSED");
        }


        if(participantDTO.getRegistrationDate().isBefore(event.getEventDate().minusDays(2))) {
            throw new EventRegistrationException("Service.REGISTRATION_CLOSED");
        }


        Participant part = new Participant();
        part.setEmailId(participantDTO.getEmailId());
        part.setGender(participantDTO.getGender());
        part.setName(participantDTO.getName());
        part.setParticipantId(participantDTO.getParticipantId());
        part.setRegistrationDate(participantDTO.getRegistrationDate());

        part.setEvent(event);

        Participant part1 = participantRepository.save(part);
        return part1.getParticipantId();
    }

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
```

Line: 52

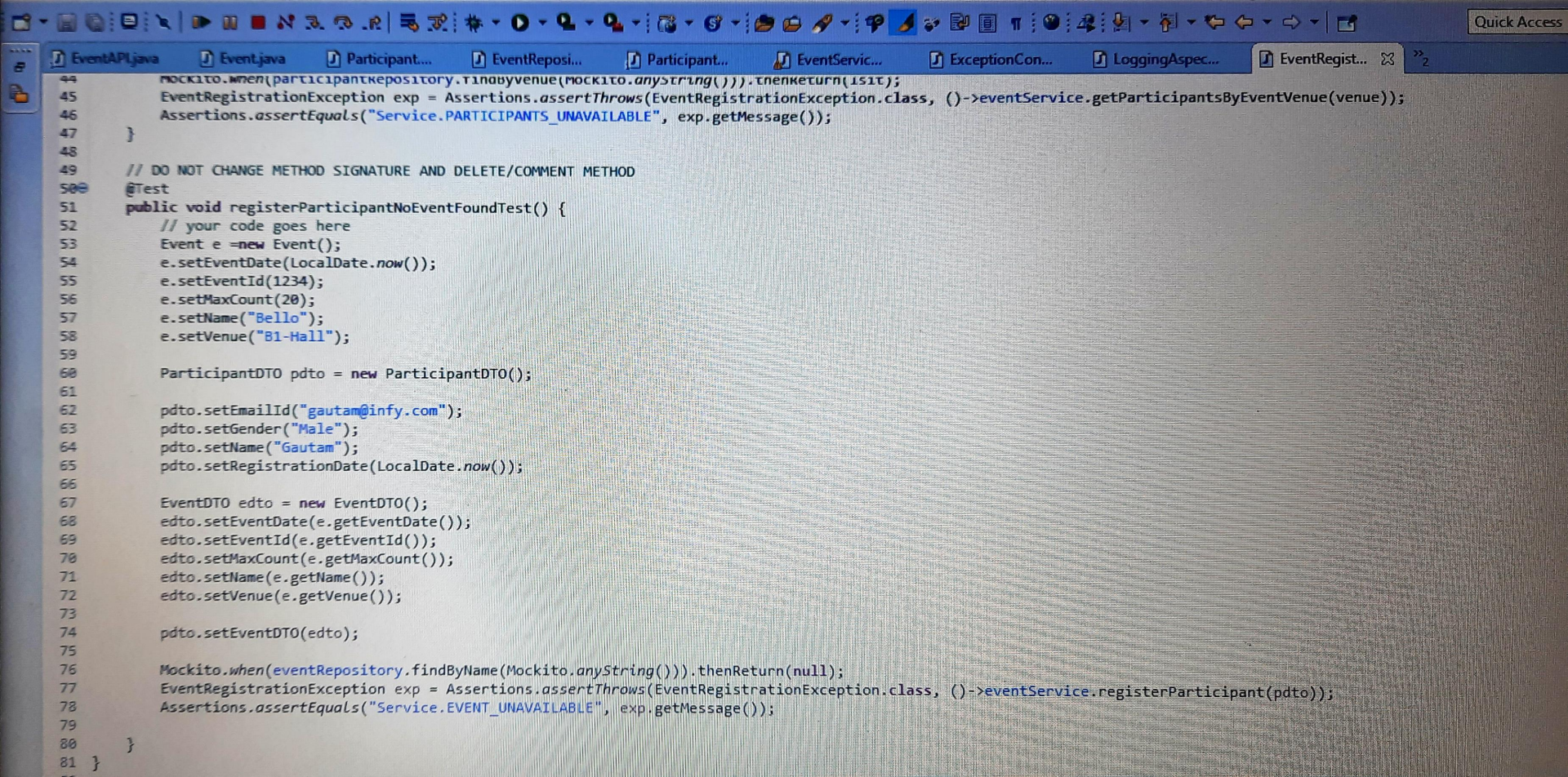
Writable | Smart Insert | 1 : 1 : 0

```java
    }

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @Override
    public List<ParticipantDTO> getParticipantsByEventVenue(String venue) throws EventRegistrationException {
        // your code goes here
//      return null;
        List<Participant> opt1 = participantRepository.findByVenue(venue);

        if(opt1.isEmpty()) {
            throw new EventRegistrationException("Service.PARTICIPANTS_UNAVAILABLE");
        }

        List<ParticipantDTO>list = new LinkedList<>();
        for(Participant p:opt1) {

            ParticipantDTO pDTO = new ParticipantDTO();
            pDTO.setEmailId(p.getEmailId());
            pDTO.setGender(p.getGender());
            pDTO.setName(p.getName());
            pDTO.setParticipantId(p.getParticipantId());
            pDTO.setRegistrationDate(p.getRegistrationDate());

            EventDTO event = new EventDTO();
            event.setEventDate(p.getEvent().getEventDate());
            event.setEventId(p.getEvent().getEventId());
            event.setMaxCount(p.getEvent().getMaxCount());
            event.setName(p.getEvent().getName());
            event.setVenue(p.getEvent().getVenue());

            pDTO.setEventDTO(event);
            list.add(pDTO);
        }
//      Collections.sort(List<ParticipantDTO> list);
//      list.sort((p1,p2)->p2.getRegistrationDate().compareTo(p1.getRegistrationDate()));
        return list;
    }
}
```

```java
package com.infy.eventregistration.utility;

import java.util.stream.Collectors;

@RestControllerAdvice
public class ExceptionControllerAdvice {

    private static final Log LOGGER = LogFactory.getLog(ExceptionControllerAdvice.class);

    @Autowired
    private Environment environment;

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @ExceptionHandler(EventRegistrationException.class)
    public ResponseEntity<ErrorInfo> eventRegistrationExceptionHandler(EventRegistrationException exception) {
        LOGGER.error(exception.getMessage(), exception);
        ErrorInfo errorInfo = new ErrorInfo();
        errorInfo.setErrorCode(HttpStatus.BAD_REQUEST.value());
        errorInfo.setErrorMessage(environment.getProperty(exception.getMessage()));
        return new ResponseEntity<>(errorInfo, HttpStatus.BAD_REQUEST);
    }

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorInfo> generalExceptionHandler(Exception exception) {
        LOGGER.error(exception.getMessage(), exception);
        ErrorInfo errorInfo = new ErrorInfo();
        errorInfo.setErrorCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
        errorInfo.setErrorMessage(environment.getProperty("General.EXCEPTION_MESSAGE"));
        return new ResponseEntity<>(errorInfo, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @ExceptionHandler({MethodArgumentNotValidException.class,ConstraintViolationException.class})
    public ResponseEntity<ErrorInfo> validatorExceptionHandler(Exception exception) {
        LOGGER.error(exception.getMessage(), exception);
        String errorMsg;
        if (exception instanceof MethodArgumentNotValidException) {
            MethodArgumentNotValidException manvException = (MethodArgumentNotValidException) exception;
```

```java
package com.infy.eventregistration.utility;

import org.apache.commons.logging.Log;

@Component
@Aspect
public class LoggingAspect {

    private static final Log LOGGER = LogFactory.getLog(LoggingAspect.class);


    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD

    @AfterThrowing(pointcut = "execution(* com.infy.eventregistration.service.*Impl.*(..))"
            ,throwing = "exception")
    public void logServiceException(EventRegistrationException exception) {
        // your code goes here
        LOGGER.error(exception.getMessage(),exception);
    }

}
```

```java

@SpringBootTest
public class EventRegistrationApplicationTests {


    @Mock
    private EventRepository eventRepository;
    @Mock
    private ParticipantRepository participantRepository;
    @InjectMocks
    private EventService eventService = new EventServiceImpl();

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @Test
    public void getParticipantsByEventVenueNoParticipantsFoundTest() {
        // your code goes here
        List<Participant> lsit = new ArrayList<>();
        String venue = "A1-HALL";

        Mockito.when(participantRepository.findByVenue(Mockito.anyString())).thenReturn(lsit);
        EventRegistrationException exp = Assertions.assertThrows(EventRegistrationException.class, ()->eventService.getParticipantsByEventVenue(venue));
        Assertions.assertEquals("Service.PARTICIPANTS_UNAVAILABLE", exp.getMessage());
    }

    // DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
    @Test
    public void registerParticipantNoEventFoundTest() {
        // your code goes here
        Event e =new Event();
        e.setEventDate(LocalDate.now());
        e.setEventId(1234);
        e.setMaxCount(20);
        e.setName("Bello");
        e.setVenue("B1-Hall");

        ParticipantDTO pdto = new ParticipantDTO();
```

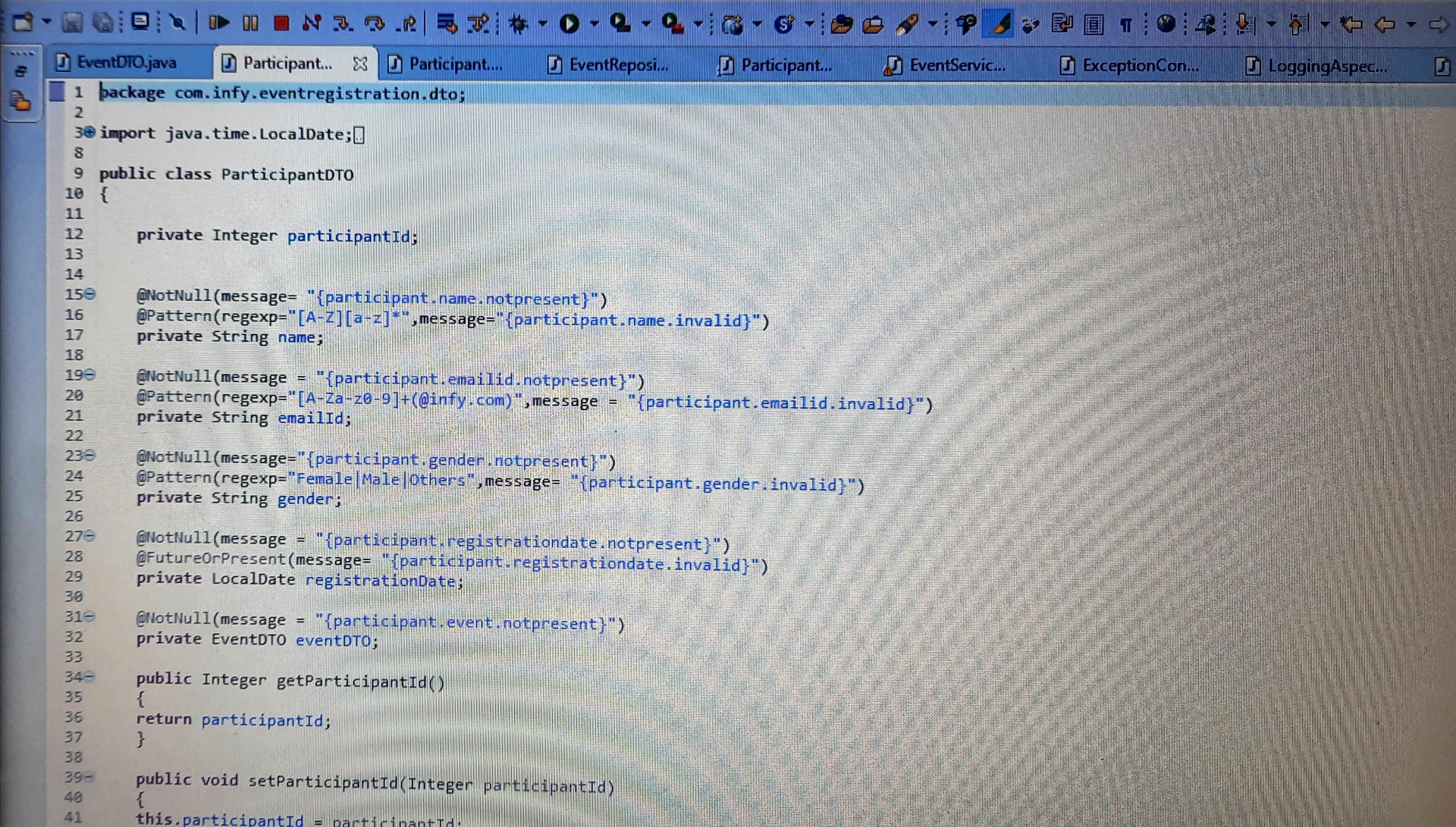

Quick Access
EventAPI.java
Event.java
Participant....
EventReposi...
Participant...
EventServic...
ExceptionCon...
LoggingAspec...
EventRegist...
Mockito.when(participantRepository.findByVenue(Mockito.anyString())).thenReturn(list);
EventRegistrationException exp = Assertions.assertThrows(EventRegistrationException.class, ()->eventService.getParticipantsByEventVenue(venue));
Assertions.assertEquals("Service.PARTICIPANTS_UNAVAILABLE", exp.getMessage());
}
// DO NOT CHANGE METHOD SIGNATURE AND DELETE/COMMENT METHOD
@Test
public void registerParticipantNoEventFoundTest() {
// your code goes here
Event e =new Event();
e.setEventDate(LocalDate.now());
e.setEventId(1234);
e.setMaxCount(20);
e.setName("Bello");
e.setVenue("B1-Hall");
ParticipantDTO pdto = new ParticipantDTO();
pdto.setEmailId("gautam@infy.com");
pdto.setGender("Male");
pdto.setName("Gautam");
pdto.setRegistrationDate(LocalDate.now());
EventDTO edto = new EventDTO();
edto.setEventDate(e.getEventDate());
edto.setEventId(e.getEventId());
edto.setMaxCount(e.getMaxCount());
edto.setName(e.getName());
edto.setVenue(e.getVenue());
pdto.setEventDTO(edto);
Mockito.when(eventRepository.findByName(Mockito.anyString())).thenReturn(null);
EventRegistrationException exp = Assertions.assertThrows(EventRegistrationException.class, ()->eventService.registerParticipant(pdto));
Assertions.assertEquals("Service.EVENT_UNAVAILABLE", exp.getMessage());
}
}

Fa_mock - EventRegistration_ToTrainee/src/main/java/com/infy/eventregistration/dto/EventDTO.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

EventDTO.java | Event.java | Participant... | EventReposi... | Participant... | EventServic... | ExceptionCon... | LoggingAspec...

```java
package com.infy.eventregistration.dto;

import java.time.LocalDate;

public class EventDTO
{
    private Integer eventId;

    @Pattern(regexp="[A-Z][0-9](-Hall)+",message = "{event.venue.invalid}")
    private String name;
    private LocalDate eventDate;
    private String venue;
    private Integer maxCount;

    public Integer getEventId()
    {
    return eventId;
    }

    public void setEventId(Integer eventId)
    {
    this.eventId = eventId;
    }

    public String getName()
    {
    return name;
    }

```



EventDTO.java | Participant... | Participant.... | EventReposi... | Participant... | EventServic... | ExceptionCon... | LoggingAspec...

```java
package com.infy.eventregistration.dto;

import java.time.LocalDate;

public class ParticipantDTO
{

    private Integer participantId;


    @NotNull(message= "{participant.name.notpresent}")
    @Pattern(regexp="[A-Z][a-z]*",message="{participant.name.invalid}")
    private String name;

    @NotNull(message = "{participant.emailid.notpresent}")
    @Pattern(regexp="[A-Za-z0-9]+(@infy.com)",message = "{participant.emailid.invalid}")
    private String emailId;

    @NotNull(message="{participant.gender.notpresent}")
    @Pattern(regexp="Female|Male|Others",message= "{participant.gender.invalid}")
    private String gender;

    @NotNull(message = "{participant.registrationdate.notpresent}")
    @FutureOrPresent(message= "{participant.registrationdate.invalid}")
    private LocalDate registrationDate;

    @NotNull(message = "{participant.event.notpresent}")
    private EventDTO eventDTO;

    public Integer getParticipantId()
    {
    return participantId;
    }

    public void setParticipantId(Integer participantId)
    {
    this.participantId = participantId;
```