УНИВЕРСИТЕТ ИТМО

Высокопроизводительные системы

Огурцов Андрей Юрьевич
Греков Марк Дмитриевич
2022 год

spring initializr

Project
- (●) Maven Project
- ( ) Gradle Project

Language
- (●) Java
- ( ) Kotlin
- ( ) Groovy

Spring Boot
- ( ) 3.0.0 (SNAPSHOT)
- ( ) 3.0.0 (M4)
- ( ) 2.7.4 (SNAPSHOT)
- (●) 2.7.3
- ( ) 2.6.12 (SNAPSHOT)
- ( ) 2.6.11

Project Metadata

Group	ru.itmo
Artifact	demoproject
Name	demoproject
Description	Demo project for Spring Boot
Package name	ru.itmo.demoproject
Packaging	(●) Jar ( ) War
Java	( ) 18 (●) 17 ( ) 11 ( ) 8

Dependencies — ADD DEPENDENCIES... CTRL + B

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

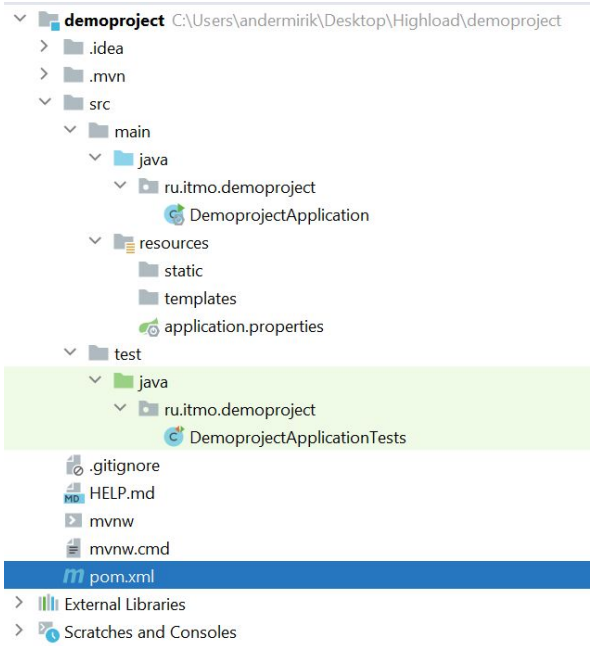
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

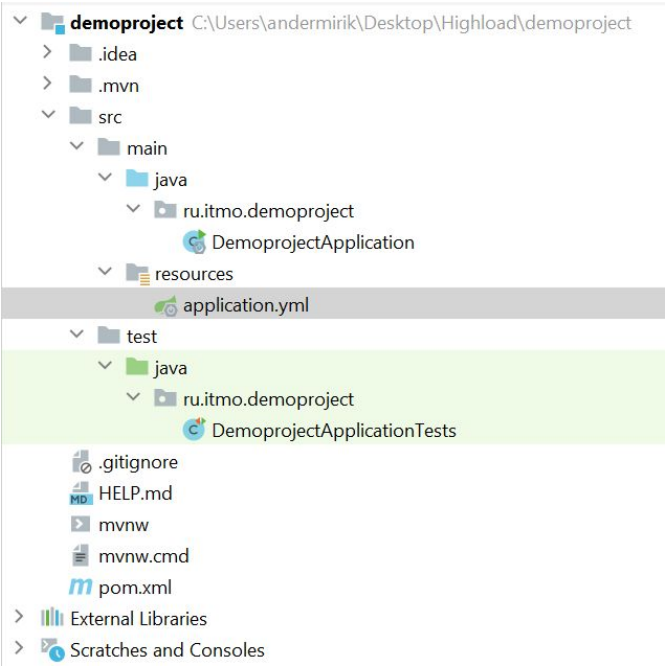
PostgreSQL Driver SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

УНИВЕРСИТЕТ ИТМО

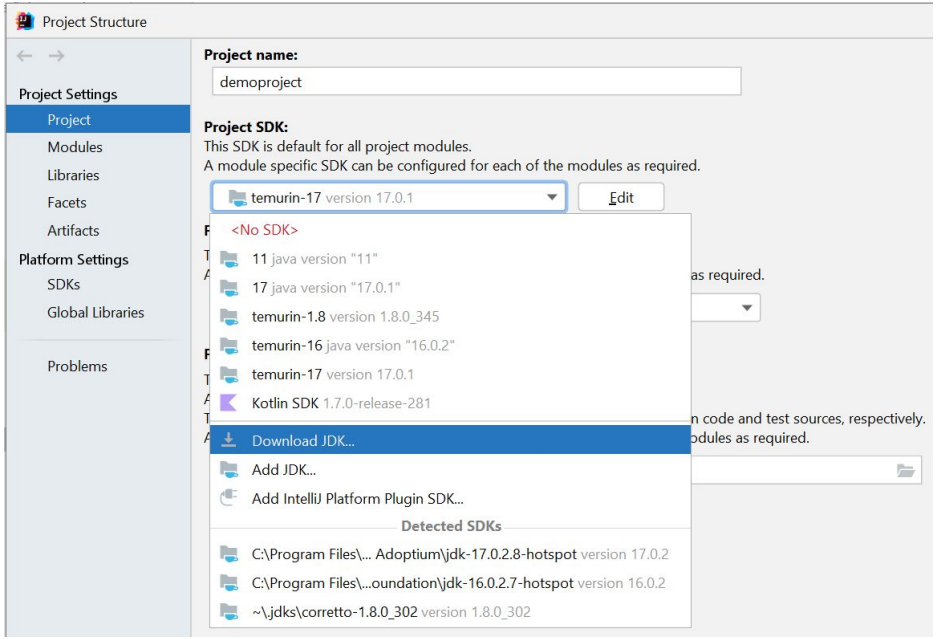
Что получим на выходе?

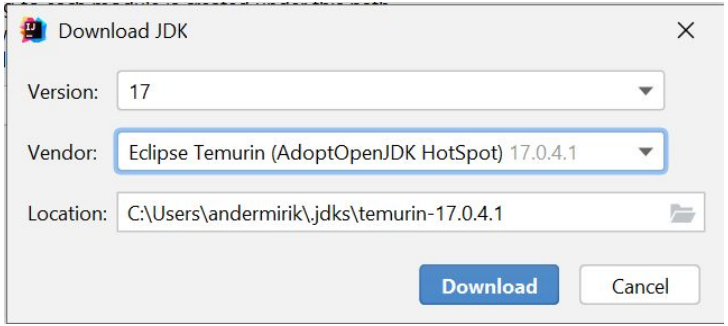




ITMOre than a UNIVERSITY

Где взять JDK?





ХТО ТАКОЙ ЭТОТ ВАШ JDK?

ITMOre than a UNIVERSITY

2536

A JavaBean is just a standard. It is a regular Java `class`, except it follows certain conventions:

1. All properties are private (use getters/setters)
2. A public no-argument constructor
3. Implements `Serializable`.

That's it. It's just a convention. Lots of libraries depend on it though.

With respect to `Serializable`, from the API documentation:

> Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

In other words, serializable objects can be written to streams, and hence files, object databases, anything really.

Also, there is no syntactic difference between a JavaBean and another class -- a class is a JavaBean if it follows the standards.

There is a term for it, because the standard allows libraries to programmatically do things with class instances you define in a predefined way. For example, if a library wants to stream any object you pass into it, it knows it can because your object is serializable (assuming the library requires your objects be proper JavaBeans).

Share Edit Follow Flag

edited Jul 6 at 14:15
Ian Boyd
237k • 241 • 844 • 1170

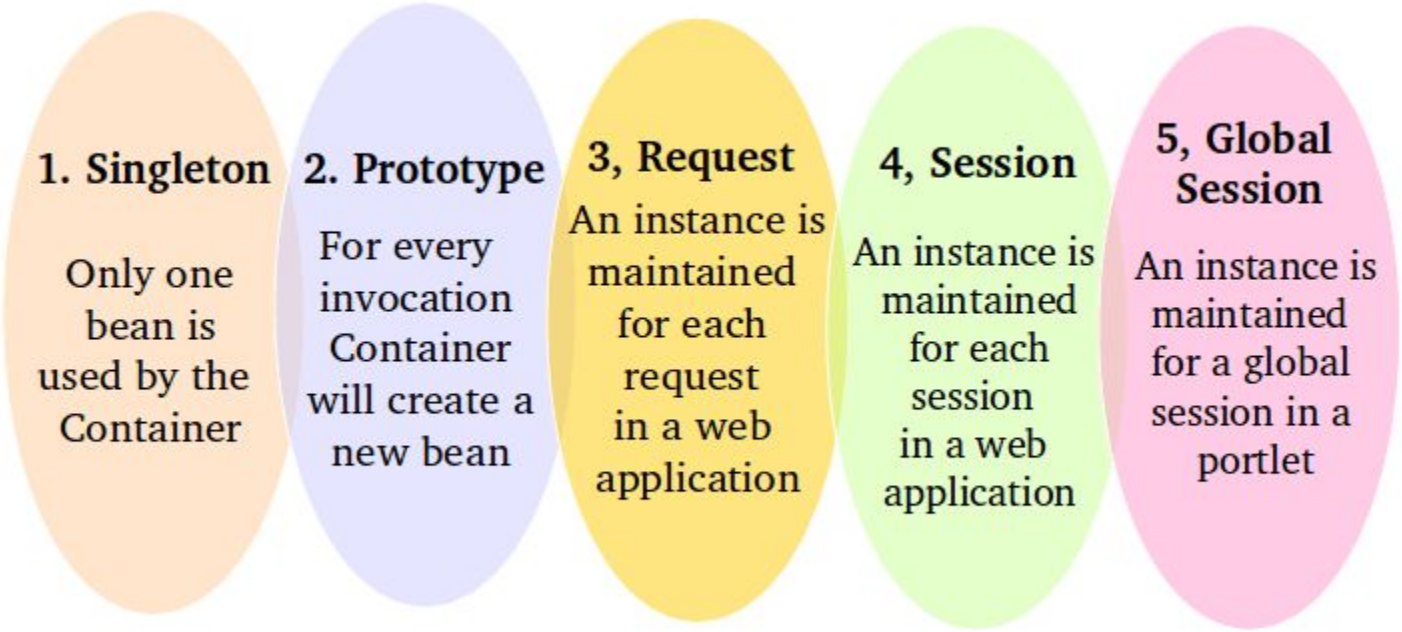
answered Jul 21, 2010 at 0:45
hvgotcodes
116k • 29 • 202 • 234

УНИВЕРСИТЕТ ИТМО

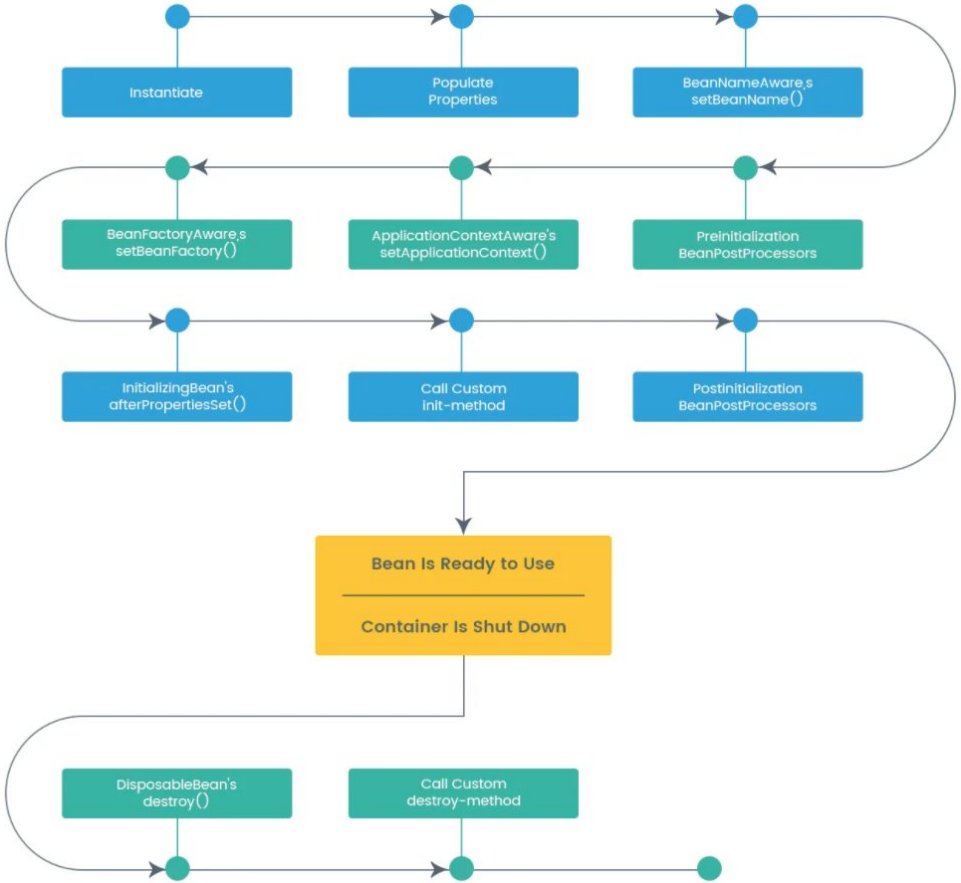
JavaBean

```
@Configuration
class CommonConfig {
  @Primary
  @Bean(name = ["sberRestTemplate"])
  fun sberRestTemplate(): RestTemplate {
    val httpRequestFactory = HttpComponentsClientHttpRequestFactory()
    httpRequestFactory.setConnectionRequestTimeout(10000)
    httpRequestFactory.setConnectTimeout(10000)
    httpRequestFactory.setReadTimeout(10000)
    return RestTemplate(httpRequestFactory)
  }

  @Bean(name = ["datascreenRestTemplate"])
  fun datascreenRestTemplate(): RestTemplate {
    val httpRequestFactory = HttpComponentsClientHttpRequestFactory()
    httpRequestFactory.setConnectionRequestTimeout(30000)
    httpRequestFactory.setConnectTimeout(30000)
    httpRequestFactory.setReadTimeout(30000)
    return RestTemplate(httpRequestFactory)
  }
}
```

Bean scope


1. Singleton
Only one bean is used by the Container
2. Prototype
For every invocation Container will create a new bean
3, Request
An instance is maintained for each request in a web application
4, Session
An instance is maintained for each session in a web application
5, Global Session
An instance is maintained for a global session in a portlet

Bean lifecycle



Bean lifecycle

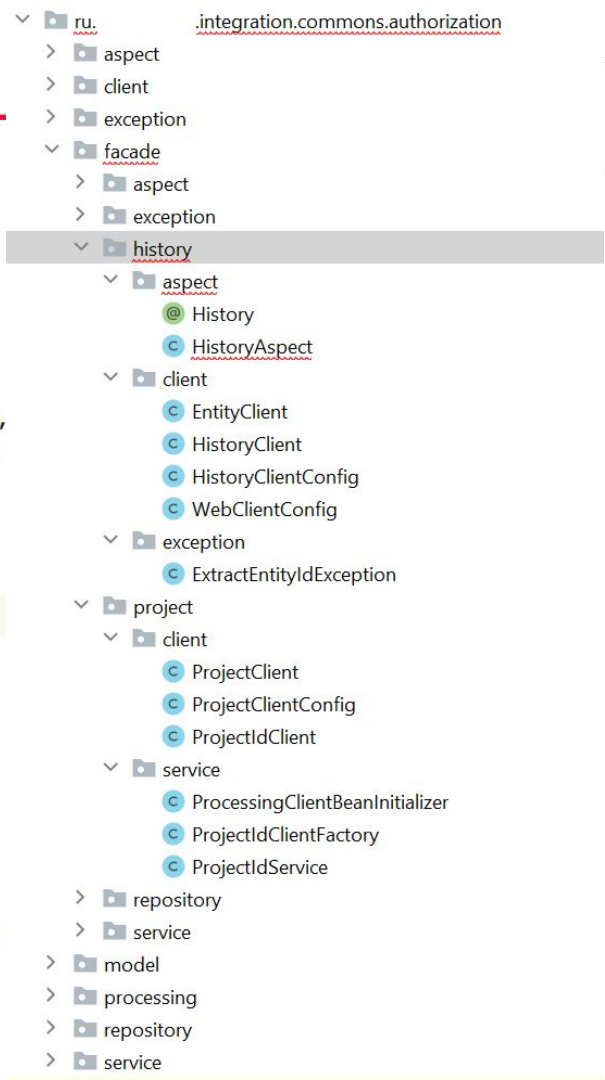
```java
@Slf4j
@Component
public class DungeonComponent {

    private List<String> masterPhrases;

    void loadFromSource() {
        masterPhrases = List.of();
        throw new UnsupportedOperationException("load is unsupported!");
    }

    @PostConstruct
    void postConstruct() {
        loadFromSource();
        log.info("dungeon constructed");
    }

    @PreDestroy
    void preDestroy() {
        masterPhrases.forEach(log::info);
        log.info("dungeon destroyed");
    }
}
```

Component Scan

```
6  @SpringBootApplication
7  public class DemoprojectApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(DemoprojectApplication.class, args);
11     }
12
13 }
```

Component Scan

```
37  @Slf4j
38  @Aspect
39  @Component
40  @ComponentScan(basePackages = {
41          "ru.            .integration.commons.authorization.facade.project",
42          "ru.            .integration.commons.authorization.facade.history"
43  })
44  @RequiredArgsConstructor
45  public class HistoryAspect {
46
47      private final Tracer tracer;
48      private final EntityClient entityClient;
49      private final HistoryClient historyClient;
50      private final ProjectIdService projectIdService;
51
52      @SneakyThrows
53      @Around("execution(public * * (..)) && @annotation(history)")
54      public Object around(ProceedingJoinPoint joinPoint, History history) {
55          log.trace("----------------------------------");
56          log.trace("around history aspect call");
57          log.trace("----------------------------------");
```



Component vs Service vs Repository

> **@Component**

This is a general-purpose stereotype annotation indicating that the class is a spring component.

What's special about @Component

`<context:component-scan>` only scans `@Component` and does not look for `@Controller`, `@Service` and `@Repository` in general. They are scanned because they themselves are annotated with `@Component`.

Thus, it's not wrong to say that `@Controller`, `@Service` and `@Repository` are special types of `@Component` annotation. `<context:component-scan>` picks them up and registers their following classes as beans, just as if they were annotated with `@Component`.

Special type annotations are also scanned, because they themselves are annotated with `@Component` annotation, which means they are also `@Component`s. If we define our own custom annotation and annotate it with `@Component`, it will also get scanned with `<context:component-scan>`

Just take a look at `@Controller`, `@Service` and `@Repository` annotation definitions:

```
@Component
public @interface Service {
    ….
}
```

```
@Component
public @interface Repository {
    ….
}
```

```
@Component
public @interface Controller {
    …
}
```

УНИВЕРСИТЕТ ИТМО

@Configuration

```
@Target(value=TYPE)
 @Retention(value=RUNTIME)
 @Documented
 @Component
public @interface Configuration
```

Indicates that a class declares one or more `@Bean` methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime, for example:

```
@Configuration
public class AppConfig {

    @Bean
    public MyBean myBean() {
        // instantiate, configure and return bean ...
    }
}
```

```
@Configuration
class CommonConfig {
  @Primary
  @Bean(name = ["sberRestTemplate"])
  fun sberRestTemplate(): RestTemplate {
    val httpRequestFactory = HttpComponentsClientHttpRequestFactory()
    httpRequestFactory.setConnectionRequestTimeout(10000)
    httpRequestFactory.setConnectTimeout(10000)
    httpRequestFactory.setReadTimeout(10000)
    return RestTemplate(httpRequestFactory)
  }

  @Bean(name = ["datascreenRestTemplate"])
  fun datascreenRestTemplate(): RestTemplate {
    val httpRequestFactory = HttpComponentsClientHttpRequestFactory()
    httpRequestFactory.setConnectionRequestTimeout(30000)
    httpRequestFactory.setConnectTimeout(30000)
    httpRequestFactory.setReadTimeout(30000)
    return RestTemplate(httpRequestFactory)
  }
}
```

Qualifier vs Primary

```
18  @Repository
19  class DatascreenRepository(
20    @Qualifier("datascreenRestTemplate")
21    private val restTemplate: RestTemplate,
22  ) {
23    private val log = LoggerFactory.getLogger(this.javaClass)
```

```
15  @Service
16  class ImageReuploadService(
17    private val sberRestTemplate: RestTemplate
18  ) {
19
20    private val log = LoggerFactory.getLogger(this.javaClass)
```

Qualifier

```
15  @RestController("projectUserControllerV2")
16  @RequestMapping("/api/v2/project")
17  public class ProjectUserController {
18
19      private final ProjectUserService service;
20
21      public ProjectUserController(@Qualifier("projectUserServiceV2") ProjectUserService service) {
22          this.service = service;
23      }
```

Conditional

```
8   @Component
9   @Conditional(ConsumerEnableCondition::class)
10  class PostgresChangePublicationOffsetTask(
11    private val jdbcTemplate: JdbcTemplate
12  ) {
```

```
8   @Component
9   class ConsumerEnableCondition : Condition {
10
11    override fun matches(context: ConditionContext, metadata: AnnotatedTypeMetadata): Boolean {
12      val beanName = context.beanFactory!!
13        .getBeanNamesForAnnotation(EnableConsuming::class.java)
14        .firstOrNull()
15
16      return beanName != null
17    }
18  }
```

УНИВЕРСИТЕТ ИТМО

@Autowired

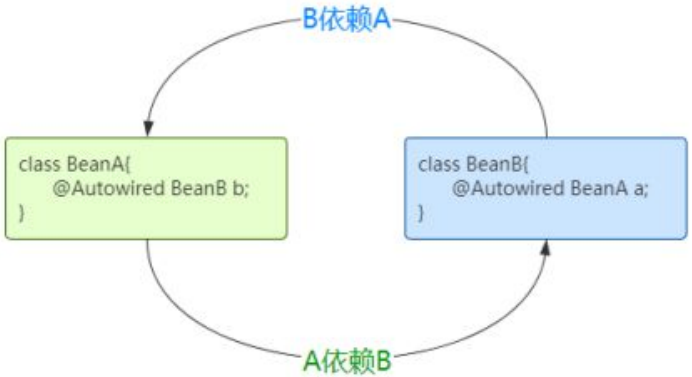
```
13  class ArtistActionTest : AbstractTest() {
14
15      @Autowired
16      private lateinit var artistAction: ArtistAction
17
18      @Autowired
19      private lateinit var messageTranslationService: MessageTranslationService
20
21      @Test
22      fun createArtistNotExistsError() {
23          val error = assertThrows<IllegalStateException> {
24              artistAction.assignArtist(zvukId = 0, user = user)
25          }
26          assertEquals(messageTranslationService.byCode( code: "ARTIST_NOT_FOUND_BY_ZVUK_ID"), error.message)
27      }
28
29      @Test
30      fun assignAssignedArtistError() {
31          val error = assertThrows<IllegalStateException> {
32              artistAction.assignArtist(zvukId = 1, user = user)
33              artistAction.assignArtist(zvukId = 1, user = user)
34          }
35          assertEquals(messageTranslationService.byCode( code: "ARTIST_ALREADY_ASSIGNED"), error.message)
36      }
```

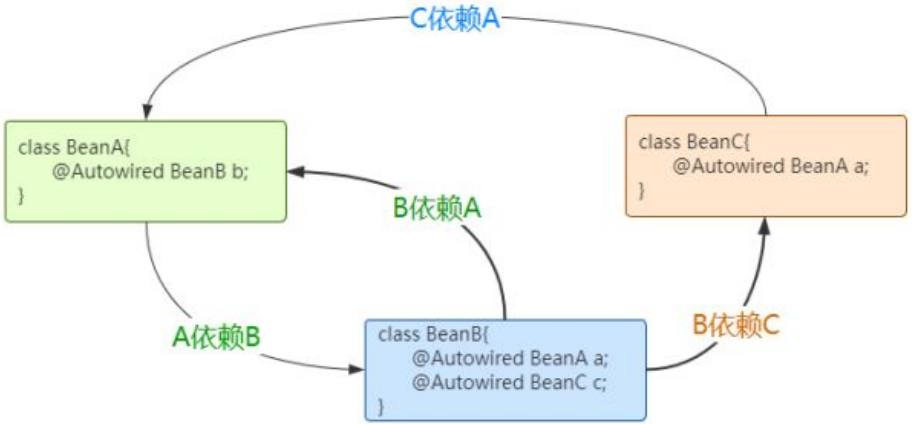
УНИВЕРСИТЕТ ИТМО

Constructor injection

```
20  @Service
21  class ReleaseService(
22    val messageTranslationService: MessageTranslationService,
23    val releaseRepository: ReleaseRepository,
24    val releaseMapper: ReleaseMapper,
25    val postgresStorage: PostgresStorage,
26    val artistService: ArtistService
27  ) {
28
29    /**
30     * Создать релиз, установить type NEW, проверить права на артиста.
31     */
32    fun create(release: ReleaseInDto, user: OptionalUser): SaveResult {...}
60
61    fun setStatus(releaseId: String, status: ReleaseStatus, comment: String, user: OptionalUser) {
62      val data = Json.json( key: "status", status.toString())
63        .set("comment", comment)
64
65      postgresStorage.save(EntityName.RELEASE, releaseId, data, user).dbRecord
66        ?: error(messageTranslationService.byCode( code: "RELEASE_NOT_EXISTS"))
67    }
```

Circular Reference





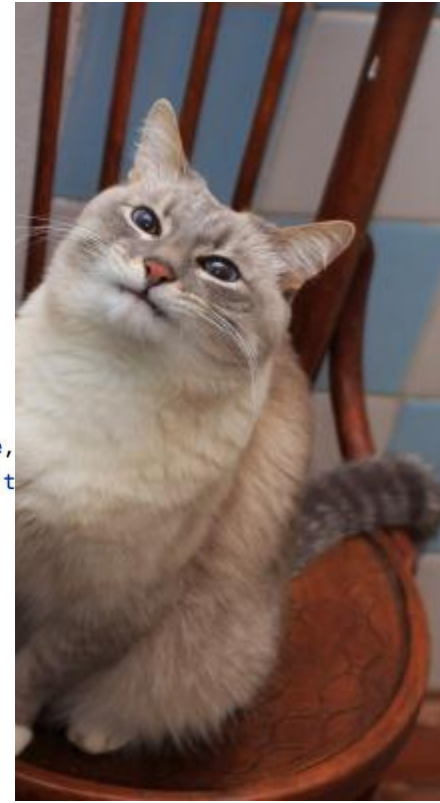
Пряник

Википедия

Мучное кондитерское изделие, выпекаемое из специального пряничного теста; печенье на меду или сахаре с пряностями. Для вкуса могут добавлять орехи, цукаты, изюм, фруктовое или ягодное повидло. На вид пряник, чаще всего, - слегка выпуклая в середине пластина прямоугольной, круглой или овальной форм, на верхней части обычно выполнены надпись или несложный рисунок, часто сверху нанесён слой кондитерской сахарной глазури.

```java
@RestController
@RequestMapping("/api/tasks")
@RequiredArgsConstructor
public class TaskResource {

    1 usage
    private final TaskService service;


    @GetMapping
    @ApiOperation(value = "Получить таски по визиту и группе")
    @ApiResponses(
            value = {
                    @ApiResponse(code = 200, message = "Операция выполнена успешна.", response = Task.class),
                    @ApiResponse(code = 204, message = "Не удалось получить лот."),
                    @ApiResponse(code = 403, message = "Ошибка авторизации."),
                    @ApiResponse(code = 404, message = "Запрошенный ресурс не существует."),
            })
    @ApiImplicitParams(
            value = {
                    @ApiImplicitParam(name = "visitId", paramType = "path", dataTypeClass = Long.class, required = true,
                    @ApiImplicitParam(name = "taskGroupId", paramType = "path", dataTypeClass = Long.class, required = t
            })
    public ResponseEntity<List<Task>> findByVisitIdAndTaskId(@RequestParam(value = "visitId") Long visitId,
                                                             @RequestParam(value = "taskGroupId") Long taskGroupId) {
        return service.findByVisitIdAndTaskGroupId(visitId, taskGroupId) Optional<List<Task>>
                .filter(list -> !list.isEmpty())
                .map(ResponseEntity::ok) Optional<ResponseEntity<List<Task>>>
                .orElse(ResponseEntity.noContent().build());
    }
}
```

УНИВЕРСИТЕТ ИТМО



Пример контроллера @PathVariable

УНИВЕРСИТЕТ ИТМО

```
@GetMapping("/{id}")
ResponseEntity<ProductEntity> findById(@PathVariable Long id) {
    return productService.findById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.noContent().build());
}

@PostMapping("/{id}/push")
ResponseEntity<ProductEntity> push(@PathVariable Long id, Long count) {
    return ResponseEntity.ok(
            productService.push(id, count)
    );
}
```

ITMO more than a UNIVERSITY

Пример контроллера @RequestBody

УНИВЕРСИТЕТ ИТМО

```java
@PostMapping
ResponseEntity<ProductEntity> save(@RequestBody ProductEntity productEntity ) {
    return ResponseEntity.ok(
            productService.save(productEntity)
    );
}
```

Плохой пример!!!

Не передавайте **@Entity** в **@RequestBody** контроллера.

Вместо этого следует использовать **DTO**,
Который только косвенно связан с данными. (об этом позже)

```java
@Data
@Entity
public class ProductEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO )
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "photo")
    private String photo;

    @Column(name = "count")
    private Long count;

    @Enumerated(EnumType.STRING)
    @Column(name = "category")
    private ProductCategory category;

    @JsonIgnore
    @OneToMany(mappedBy = "recipe")
    private List<RecipeProductEntity> recipes;

}
```

Пример контроллера @RequestParam

УНИВЕРСИТЕТ ИТМО

В запросе **@RequestParam** с типом **List<Long>**
будут выглядеть, как **ids[]=1,2,3,4,5**

```
@GetMapping("/by-ids")
ResponseEntity<List<ProductEntity>> findByIds(@RequestParam(name = "ids[]") List<Long> ids) {
    return ResponseEntity.ok(
            productService.findByIds(ids)
    );
}
```

@RequestParam могут не передаваться в запросе.
В таком случае мы указываем **required=false**,
а в переменную будет помещено значение **null**,
если Spring не сможет найти ожидаемый параметр в запросе.

ITMO more than a UNIVERSITY

Пример контроллера на скачивание файла

УНИВЕРСИТЕТ ИТМО

```
78      @PostMapping("/pdf")
79      @ApiOperation(value = "Скачать отчет по чекам в pdf", produces = "application/octet-stream")
80      public ResponseEntity<Void> downloadReportChecksPdf(@RequestHeader("Authorization") String token,
81                                                          @RequestBody Map<String, String> filter,
82                                                          HttpServletResponse response) {
83          final OutputStream pdfos = service.downloadPdfChecksReport(token, filter);
84          try (
85                  ByteArrayInputStream bis = new ByteArrayInputStream(((ByteArrayOutputStream) pdfos).toByteArray());
86                  OutputStream os = response.getOutputStream()
87          ) {
88              String filename = "Чеки.zip";
89              response.setHeader( s: "Content-Type",  s1: "application/zip");
90              ContentDisposition contentDisposition = ContentDisposition.builder( type: "attachment")
91                      .filename(filename, StandardCharsets.UTF_8)
92                      .build();
93              response.setHeader( s: "Content-Disposition", contentDisposition.toString());
94              bis.transferTo(os);
95              response.flushBuffer();
96              return ResponseEntity.status(HttpStatus.OK).build();
97          } catch (Exception ex) {
98              log.error("download pdf exception was thrown", ex);
99              return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
100         }
101     }
```

Скачивание файлов при помощи контроллера

УНИВЕРСИТЕТ ИТМО

Для того чтобы вернуть файл из контроллера не обязательно работать с **HttpServletResponse** напрямую.

Можно воспользоваться одним из следующих возвращаемых типов:

- **ResponseEntity<Resource>** или
- **ResponseEntity<InputStreamResource>** или
- **ResponseEntity<byte[]>**

ITMO more than a UNIVERSITY

УНИВЕРСИТЕТ ИТМО

Пример контроллера на загрузку файла

```
@PostMapping("/upload")
@ApiOperation(value = "Загрузка фотографии с нанесением даты", consumes = "multipart/form-data")
public ResponseEntity<PhotoUploadResult> upload(@RequestParam("taskId") Long taskId,
                                                @RequestParam("mobileId") String mobileId,
                                                @RequestPart(value = "photo", required = true) MultipartFile photo){
    return photoService.uploadPhoto(mobileId, taskId, photo) Optional<PhotoUploadResult>
            .map(ResponseEntity::ok) Optional<ResponseEntity<PhotoUploadResult>>
            .orElse(ResponseEntity.noContent().build());
}
```

В дальнейшем из **photo** можно извлечь информацию при помощи **photo.getInputStream()**

ITMO more than a UNIVERSITY

НЕ ДЕЛАЙТЕ ТАК

```
@GetMapping
public List<Person> getAll(Model model) {
    List<Person> personList = peopleService.getAll();
    model.addAttribute("personList", personList);
    model.addAttribute("personSize", personList.size());
    return personList;
}
```

НЕ ДЕЛАЙТЕ ТАК

УНИВЕРСИТЕТ ИТМО

```
public class AbstractNotCreateWithVersionAndPackageAndTaskResource<T extends AbstractEntity<T>, ID extends Serializable> {

    protected Logger log;
    protected Class<T> clazz;

    @Autowired
    protected AbstractServiceWithVersionAndPackageAndTask<T,ID> service;

    /**
     * GET  /activities : get all the activities.
     *
     * @return the ResponseEntity with status 200 (OK) and the list of activities in body
     */

    @GetMapping
    @Timed
    public ResponseEntity<List<T>> getAll(@RequestParam(value = "storeId", required = false) ID storeId,
                                          @RequestParam(value = "taskId", required = false) Long taskId,
                                          @RequestParam(value = "ver",required = false) String version,
                                          @RequestParam(value = "package", required = false) String packageName) throws Exception {
        log.debug("REST request to get all {}",clazz.getSimpleName());
        List<T> res=service.findAll(storeId, taskId, version, packageName);
        if (res.size() > 0) {
            return new ResponseEntity<List<T>>(res, HttpStatus.OK);
        }
        else return new ResponseEntity<List<T>>(new ArrayList<>(),HttpStatus.OK);
    }
}
```

НЕ ДЕЛАЙТЕ ТАК

УНИВЕРСИТЕТ ИТМО

```
@RestController
@RequestMapping("/api/activity-transaction-result-type-info")
public class ActivityTransactionResultTypeInfoResource extends AbstractNotCreateWithVersionAndPackageAndTaskResource<ActivityTransactionResultTypeInf

    {
        log=LoggerFactory.getLogger(ActivityTransactionResultTypeInfoResource.class);
        clazz=ActivityTransactionResultTypeInfo.class;
    }

}
```


ITMO more than a UNIVERSITY

Используйте пагинацию на findAll

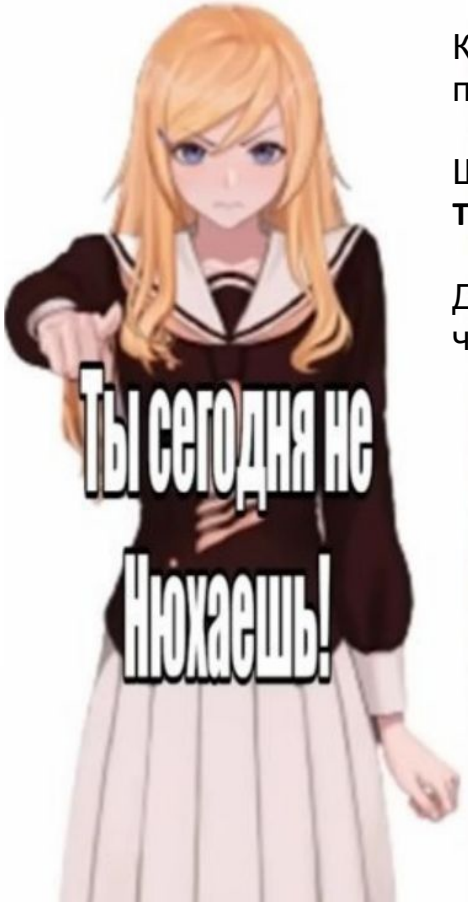
УНИВЕРСИТЕТ ИТМО

```
@PostMapping
public ResponseEntity<List<SmsPage>> findAll(@RequestBody SmsFilter filter,
                                             @RequestParam("page") Long page,
                                             @RequestParam("size") Long size) {
    return service.findAll(filter, page, size) Optional<SmsWrapper>
            .filter(sms -> !sms.getSmsPageList().isEmpty())
            .map(wrap -> ResponseEntity
                    .status(HttpStatus.OK)
                            .header( s: "x-total-count", wrap.getCount().toString())
                            .body(wrap.getSmsPageList())
                    ) Optional<ResponseEntity<List<SmsPage>>>
            .orElse(ResponseEntity.noContent().build());
}
```

ITMO more than a UNIVERSITY

Controller

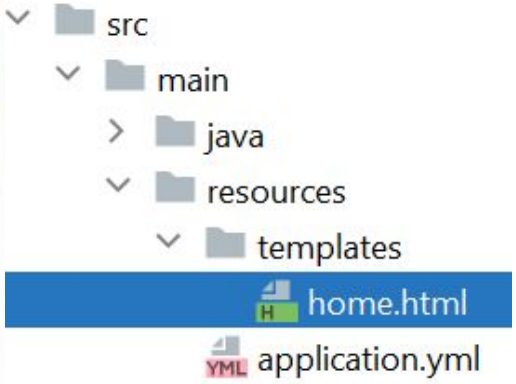
```
@Controller
@CrossOrigin("*")
public class HomeController {
    @GetMapping("/")
    public String home(Model model){
        return "home";
    }

}
```



Контроллер возвращает путь к шаблону

Шаблон рендерится при помощи **Thymeleaf**

Данные в шаблон попадают через **Model**



Valid

УНИВЕРСИТЕТ ИТМО

@Valid UserAccount userAccount;

```
public class UserAccount {

    @NotNull
    @Size(min = 4, max = 15)
    private String password;

    @NotBlank
    private String name;

    @Min(value = 18, message = "Age should not be less than 18")
    private int age;

    @NotBlank
    private String phone;

    // standard constructors / setters / getters / toString

}
```

ITMO more than a UNIVERSITY

Validated

УНИВЕРСИТЕТ ИТМО

@Validated(BasicInfo.class) UserAccount userAccount;

```
public class UserAccount {

    @NotNull(groups = BasicInfo.class)
    @Size(min = 4, max = 15, groups = BasicInfo.class)
    private String password;

    @NotBlank(groups = BasicInfo.class)
    private String name;

    @Min(value = 18, message = "Age should not be less than 18", groups = AdvanceInfo.class)
    private int age;

    @NotBlank(groups = AdvanceInfo.class)
    private String phone;

    // standard constructors / setters / getters / toString

}
```

ITMO more than a UNIVERSITY

4. Using *@Valid* Annotation to Mark Nested Objects

The *@Valid* annotation is used to mark nested attributes, in particular. This triggers the validation of the nested object. For instance, in our current scenario, we can create a *UserAddress* object:

```
public class UserAddress {

    @NotBlank
    private String countryCode;

    // standard constructors / setters / getters / toString
}
```

To ensure validation of this nested object, we'll decorate the attribute with the *@Valid* annotation:

```
public class UserAccount {

    //...

    @Valid
    @NotNull(groups = AdvanceInfo.class)
    private UserAddress useraddress;

    // standard constructors / setters / getters / toString
}
```

Filter

УНИВЕРСИТЕТ ИТМО

```
18  @Component
19  @RequiredArgsConstructor
20  public class CustomTraceFilter extends GenericFilterBean {
21
22      private final Tracer tracer;
23
24      @Override
25      @SneakyThrows
26      public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {
27          Span currentSpan = this.tracer.currentSpan();
28          if (currentSpan == null) {
29              chain.doFilter(request, response);
30              return;
31          }
32          ((HttpServletResponse) response).addHeader( s: "TRACE-ID", currentSpan.context().traceIdString());
33          String query = ((HttpServletRequest)request).getQueryString();
34          currentSpan.tag( s: "query", query==null ? "" : query);
35          chain.doFilter(request, response);
36      }
37  }
```

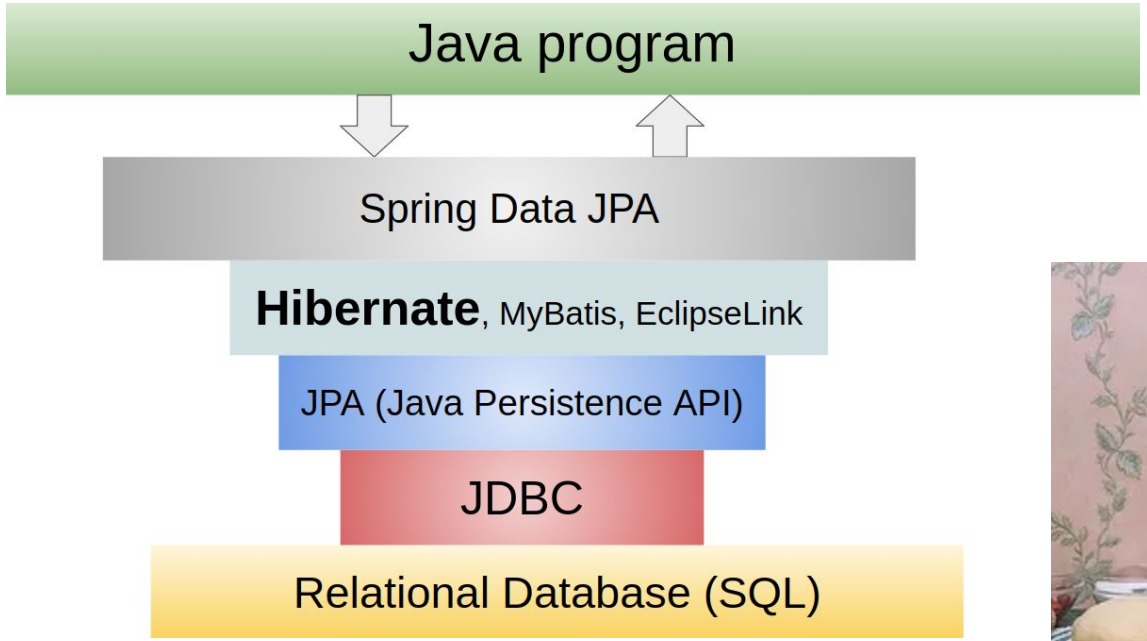
ITMOre than a UNIVERSITY

ExceptionHandler

Любой выкинутый за время работы приложения **Exception** будет перехвачен **ExceptionHandler** и сформирует ответ, который необходимо вернуть пользователю.

```
8   @ControllerAdvice
9   public class MyExceptionHandler extends ResponseEntityExceptionHandler {
10
11      @ExceptionHandler(IllegalArgumentException.class)
12  @   public ResponseEntity<String> exceptionHandler(Exception ex) {
13          return ResponseEntity.badRequest().body(ex.getMessage());
14      }
15
16  }
```

Spring Data JPA



Конфигурируем Datasource

УНИВЕРСИТЕТ ИТМО

application.yml

```yaml
server:
  port: 8080

spring:
  jpa:
    show-sql: true
    database: postgresql
    database-platform: org.hibernate.dialect.PostgreSQL10Dialect
    generate-ddl: false
    properties:
      hibernate:
        show_sql=true:
    hibernate:
      ddl-auto: update

  datasource:
    username: ${DB_USER:postgres}
    password: ${DB_PASSWORD:postgres}
    url: jdbc:postgresql://${DB_HOST:localhost}:${DB_PORT:5432}/postgres
    driver-class-name: org.postgresql.Driver

jwt:
  secret: IHATECOMPUTERSCIENCE
```

Важно чтобы была зависимость **spring-data** и драйвера **postgres**

jwt.secret – собственная переменная, которая была введена для валидации JWT

@Entity

УНИВЕРСИТЕТ ИТМО

```java
@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
@Builder
@AllArgsConstructor
public class OrderEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO )
    Long id;

    Long userId;

    @JsonFormat(pattern = "yyyy-MM-dd'T'HH:mm:ss.SSSSSS'Z'", timezone = "UTC")
    Instant timestamp;

    @Enumerated(EnumType.STRING)
    @Column(name = "order_status")
    OrderStatus orderStatus;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinColumn(name = "position_id", nullable = false)
    List<PositionEntity> positions;

    @PrePersist
    void prePersist() {
        this.timestamp = Instant.now();
    }
}
```

```java
public enum OrderStatus {
    ACCEPTED,
    NOT_READY,
    READY,
}
```

@Entity

УНИВЕРСИТЕТ ИТМО

```java
@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Entity
@Builder
@AllArgsConstructor
public class PositionEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO )
    private Long id;

    @Column(name = "title")
    private String title;

    @Column(name = "photo")
    private String photo;

    @Column(name = "description")
    private  String description;

    @Column(name="price")
    private Long price;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "recipe_id", nullable = false)
    private RecipeEntity recipe;

    private Boolean isClosed;
}
```

@Entity

УНИВЕРСИТЕТ ИТМО

```java
@Getter
@Setter
@RequiredArgsConstructor
@Entity
@Builder
@AllArgsConstructor
public class RecipeEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO )
    private Long id;

    @Column(name = "title")
    private String title;

    @Column(name = "description")
    private String description;

    @OneToMany(mappedBy = "product")
    private List<RecipeProductEntity> products;

}
```

@Entity

УНИВЕРСИТЕТ ИТМО

```java
@Entity
@Table(name = "recipe_product")
@IdClass(RecipeProductId.class)
@Getter
@Setter
public class RecipeProductEntity {

    @Id
    @ManyToOne
    @JoinColumn(name = "recipe_id", referencedColumnName = "id")
    private RecipeEntity recipe;

    @Id
    @ManyToOne
    @JoinColumn(name = "product_id", referencedColumnName = "id")
    private ProductEntity product;

    @Column(name = "count")
    private Long count;

}
```

```java
@Data
public class RecipeProductId implements Serializable {
    private Long recipe;
    private Long product;
}
```

Entity

УНИВЕРСИТЕТ ИТМО

```java
@Data
@Entity
@Builder
@Table(name="users")
@NoArgsConstructor
public class UserEntity implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO )
    private Long id;

    @Column(name="login")
    private String login;

    @JsonIgnore
    @Column(name="password")
    private String password;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "role_id", nullable = false)
    private RoleEntity role;
}
```

N+1 проблема

Debugger Console Actuator

```
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
Hibernate: select accesslist0_.role_id as role_id11_0_0_, accesslist0_.id as id1_0_0_, accesslist0_.id as id1_0_1_, a
```

N+1 как должно быть

УНИВЕРСИТЕТ ИТМО

```
Hibernate: select projectent0_.id as id1_3_, projectent0_.author_id as author_i2_3_, projectent0_.description as desc
Hibernate: select meta0_.project_id as project_4_4_1_, meta0_.id as id1_4_1_, meta0_.id as id1_4_0_, meta0_.key as ke
Hibernate: select ganttmodel0_.project_id as project_3_2_1_, ganttmodel0_.id as id1_2_1_, ganttmodel0_.id as id1_2_0_
Hibernate: select assignedus0_.project_id as project_3_1_2_, assignedus0_.id as id1_1_2_, assignedus0_.id as id1_1_1_
Hibernate: select accesslist0_.role_id as role_id11_0_1_, accesslist0_.id as id1_0_1_, accesslist0_.id as id1_0_0_, a
Hibernate: select projectent0_.id as id1_3_, projectent0_.author_id as author_i2_3_, projectent0_.description as desc
Hibernate: select meta0_.project_id as project_4_4_1_, meta0_.id as id1_4_1_, meta0_.id as id1_4_0_, meta0_.key as ke
Hibernate: select ganttmodel0_.project_id as project_3_2_1_, ganttmodel0_.id as id1_2_1_, ganttmodel0_.id as id1_2_0_
Hibernate: select assignedus0_.project_id as project_3_1_2_, assignedus0_.id as id1_1_2_, assignedus0_.id as id1_1_1_
Hibernate: select accesslist0_.role_id as role_id11_0_1_, accesslist0_.id as id1_0_1_, accesslist0_.id as id1_0_0_, a
```

ITMO more than a UNIVERSITY

N+1 проблема

УНИВЕРСИТЕТ ИТМО

```java
@Override
@Query(nativeQuery = true, value = """
    WITH RECURSIVE tree AS (
        SELECT *,
                id as root
            FROM document_template_processing_service.t_document_template_attribute
            WHERE parent_attribute_id IS NULL
        UNION ALL
            SELECT document_template_processing_service.t_document_template_attribute.*,
                tree.root
            FROM document_template_processing_service.t_document_template_attribute
            JOIN tree ON document_template_processing_service.t_document_template_attribute.parent_attribute_id = tree.id
    )
    SELECT DISTINCT dt.id, * FROM document_template_processing_service.t_document_template as dt
    LEFT JOIN tree ON dt.id = tree.root
    WHERE dt.id = :id
""")
Optional<DocumentTemplateEntity> findById(UUID id);
```

Console Actuator

```
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
2022-01-26 17:56:07+05 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_0_, attribut
```

N+1 как должно быть

УНИВЕРСИТЕТ ИТМО

```
2022-01-26 19:48:40±07 [restartedMain] [INFO] ru.sintezat.asu.processing.document.template.DocumentTemplateServiceEntrypoint: Started Docu
2022-01-26 19:48:44±07 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select documentte0_.id as id1_0_0_, documentte0_.created_at as crea
2022-01-26 19:48:44±07 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_2_, attribut
2022-01-26 19:48:44±07 [reactor-http-nio-4] [DEBUG] org.hibernate.SQL: select attributes0_.parent_attribute_id as parent_11_1_2_, attribut
```

ITMO more than a UNIVERSITY

N+1 как решить – BatchSize

УНИВЕРСИТЕТ ИТМО

DocumentAttributeEntity.java | DocumentTemplateEntity.java | DocumentTemplateRepository.java | DirectoryStor

```
27    @Enumerated(EnumType.STRING)
28    private LinkedDataType linkedDataType;
29
30    @Column(name = "linked_data_id")
31    private String linkedDataId;
32
33    @ManyToOne(cascade = {CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST,
34    @JoinColumn(name = "parent_attribute_id", referencedColumnName = "id")
35    private DocumentAttributeEntity parentAttribute;
36
37    @BatchSize(size = 500)
38    @OneToMany(
39            mappedBy = "parentAttribute",
40            orphanRemoval = true,
41            fetch = FetchType.EAGER,
42            cascade = CascadeType.ALL
43    )
44    private List<DocumentAttributeEntity> attributes = new ArrayList<>();
45
```

ITMO more than a UNIVERSITY

N+1 как решить – SUBSELECT

УНИВЕРСИТЕТ ИТМО

```
43      @OneToMany(
44              mappedBy = "documentTemplate",
45              fetch = FetchType.EAGER,
46              orphanRemoval = true,
47              cascade = CascadeType.ALL
48      )
49      @Fetch(FetchMode.SUBSELECT)
50      private List<DocumentAttributeEntity> attributes;
51  }
52
```

ITMO more than a UNIVERSITY

N+1 как решить – JOIN

УНИВЕРСИТЕТ ИТМО

```
43      @OneToMany(
44              mappedBy = "documentTemplate",
45              fetch = FetchType.EAGER,
46              orphanRemoval = true,
47              cascade = CascadeType.ALL
48      )
49      @Fetch(FetchMode.JOIN      )
50      private List<DocumentAttributeEntity> attributes;
51  }
52
```

ITMOre than a UNIVERSITY

FetchType

УНИВЕРСИТЕТ ИТМО

75

`EAGER` loading of collections means that they are fetched fully at the time their parent is fetched. So if you have `Course` and it has `List<Student>`, all the students are fetched *from the database* at the time the `Course` is fetched.

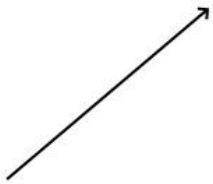
`LAZY` on the other hand means that the contents of the `List` are fetched only when you try to access them. For example, by calling `course.getStudents().iterator()`. Calling any access method on the `List` will initiate a call to the database to retrieve the elements. This is implemented by creating a Proxy around the `List` (or `Set`). So for your lazy collections, the concrete types are not `ArrayList` and `HashSet`, but `PersistentSet` and `PersistentList` (or `PersistentBag`)

FetchType.EAGER

```java
@Entity
public class Student
{
	@Id
	@GeneratedValue
	private long id;

	private String name;

	@OneToOne( fetch = FetchType.EAGER )
	private Passport passport;

}
```

```java
@Entity
public class Passport
{
	@Id
	@GeneratedValue
	private long id;

	private String number;
}
```

```java
public Student FetchTypeDemo()
{
	Student student = em.find( Student.class, 21 );
	Passport passport = student.getPassport();
	String number = passport.getNumber();
	return student;
}
```



```
Hibernate:
    select
      student0_.id as id1_3_0_,
      student0_.name as name2_3_0_,
      student0_.passport_id as passport3_3_0_,
      passport1_.id as id1_1_1_,
      passport1_.number as number2_1_1_
    from
      student student0_
    left outer join
      passport passport1_
        on student0_.passport_id=passport1_.id
    where
      student0_.id=?
```

FetchType.LAZY

```java
@Entity
public class Student
{
	@Id
	@GeneratedValue
	private long id;

	private String name;

	@OneToOne( fetch = FetchType.LAZY )
	private Passport passport;

	protected Student()
	{
	}

 }
```

```java
@Entity
public class Passport
{
	@Id
	@GeneratedValue
	private long id;

	private String number;

}
```

```java
public Student FetchTypeDemo()
{
	Student student = em.find( Student.class, 21 );
	Passport passport = student.getPassport();
	String number = passport.getNumber();
	return student;
}
```

```
Hibernate:
  select
    student0_.id as id1_3_0_,
    student0_.name as name2_3_0_,
    student0_.passport_id as passport3_3_0_
  from
    student student0_
  where
    student0_.id=?
```

```
Hibernate:
  select
    passport0_.id as id1_1_0_,
    passport0_.number as number2_1_0_
  from
    passport passport0_
  where
    passport0_.id=?
```

LazyInitializationException :

Hibernate throws the LazyInitializationException when it needs to initialize a lazily fetched association to another entity without an active session context.

That's usually the case if you try to use an uninitialize association in your client application or web layer.

Entity vs DTO

94

Short answer:

- Entities *may be* part of a business domain. Thus, they can implement behavior and be applied to different use cases within the domain.
- DTOs are used only to transfer data from one process or context to another. As such, they are without behavior - except for very basic and usually standardised storage and retrieval functions.

Long answer:

While the term "Data Transfer Object" (DTO) is defined quite unambiguously, the term "Entity" is interpreted differently in various contexts.

The most relevant interpretations of the term "Entity", in my opinion, are the following three:

1. *In the context of entity-relationship- and ORM-frameworks - specifically Enterprise Java and Jpa:*
   "An object that represents persistent data maintained in a database."
2. *In the context of "Domain-Driven Design" (by Eric Evans):*
   "An object defined primarily by its identity, rather than its attributes."
3. *In the context of "Clean Architecture" (by Robert C. Martin):*
   "An object that encapsulates enterprise-wide critical business rules."

The Jee- and Jpa-community sees entities primarily as objects mapped to a database table. This point of view is very close to the definition of a DTO - and that's where much of the confusion probably stems from.

In the context of domain-driven-design, as well as Robert Martins point of view, however, Entities are part of a business domain and thus can and should implement behavior.

Validation

УНИВЕРСИТЕТ ИТМО

- @NotNull, @NotEmpty, @NotBlank, @Null
- @Email
- @Min, @Max, @Length
- @Size
- @Future, @Past
- @Negative, @NegativeOrZero, @Positive, @PositiveOrZero

ITMO more than a UNIVERSITY

https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#section-builtin-constraints

JpaRepository

УНИВЕРСИТЕТ ИТМО

```
8  public interface PositionRepositiory extends JpaRepository<PositionEntity, Long> {
9      List<PositionEntity> findAllByIsClosedFalse();
10     List<PositionEntity> find
11 }
12
```

- findAllBy
- findBy
- findPositionEntityBy
- findPositionEntitiesBy
- findDistinctBy
- findDistinctFirstBy
- findDistinctTopBy
- findFirstBy
- findTopBy

Press Enter to insert, Tab to replace Next Tip

ITMO more than a UNIVERSITY

JpaRepository

При помощи рефлексии `JpaRepository` может составлять относительно простые запросы.

```
8  public interface PositionRepositiory extends JpaRepository<PositionEntity, Long> {
9      List<PositionEntity> findAllByIsClosedFalse();
10     List<PositionEntity> findAllByTitleLikeAndPriceBetween(String titleLike, Long a, Long b);
11 }
```

Стандартные методы:

- .save()
- .findById()
- .findAll()
- .delete()
- .deleteAll()
- .exists()

JpaRepository

УНИВЕРСИТЕТ ИТМО

Более сложные запросы следует составлять при помощи **@Query**

@Query бывает **native** и по умолчанию.

native=true обозначает использовать голый диалект SQL сервера.

Native по умолчанию использует диалект JPA

```
@Query("SELECT * FROM t_notification WHERE :userId = any(target_users) ORDER BY timestamp DESC")
List<NotificationEntity> findAllByUserId(UUID userId);
```

ITMO more than a UNIVERSITY

JpaRepository

УНИВЕРСИТЕТ ИТМО

@Query можно совместить вместе с **@Modifying**

```
16      @Modifying
17      @Query(
18              """
19                      UPDATE t_comment
20                      SET is_deleted = true, modified_at = :modifiedAt
21                      WHERE id = :id
22                      """
23      )
24      Integer deleteCommentEntity(Instant modifiedAt, UUID id);
25
        1 related problem
26      @Modifying
27      @Query(
28              """
29                      UPDATE t_comment
30                      SET text = :text, modified_at = :modifiedAt
31                      WHERE id = :id
32                      """
33      )
34      Integer editCommentEntity(String text, Instant modifiedAt, UUID id);
```

@Transactional

УНИВЕРСИТЕТ ИТМО

@Transactional заставляет метод выполняться в рамках одной транзакции. Брошенное исключение откатит транзакцию.

```
26      @Transactional
27  @   public PositionEntity save(PositionDto positionDto) {
28
29          RecipeEntity recipeEntity = recipeRepository.findById(positionDto.getRecipeId()).orElse( other: null);
30          if(recipeEntity == null) throw new IllegalArgumentException("Рецепт не найден.");
31
32          List<RecipeProductEntity> products = recipeProductRepository.findRecipeProductEntitiesByRecipeId(recipeEntity.getId());
33          for(RecipeProductEntity product : products) {
34              productService.pull(product.getProduct().getId(), product.getCount());
35          }
36
37          PositionEntity positionEntity = PositionEntity.builder()
38                  .title(positionDto.getTitle())
39                  .description(positionDto.getDescription())
40                  .photo(positionDto.getPhoto())
41                  .price(positionDto.getPrice())
42                  .recipe(recipeEntity)
43                  .isClosed(false)
44                  .build();
45
46          return positionRepositiory.save(positionEntity);
47      }
```

Класс *Report*:

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Report {
    @Id
    private long id;
    private boolean published;
}
```

УНИВЕРСИТЕТ ИТМО

ReportRepository

Метод репозитория *updatePublished()* пусть будет *@Transactional*, так как иногда нам надо вызывать его отдельно (вне отправки отчета):

```java
@Transactional
@Repository
public interface ReportRepository extends CrudRepository<Report, Long> {

      @Modifying
      @Query("update report set published = 'true' where id=:id")
      void updatePublished(long id);

}
```

ITMOre than a UNIVERSITY

https://sysout.ru/transaction-propagation/

ReportService

УНИВЕРСИТЕТ ИТМО

Метод сервиса отправляет отчет и если все ок, вызывает метод репозитория.

Для простоты пусть отправка отчета заключается в выводе его в консоль.

Пометим метод тоже как *@Transactional*, потому что все должно быть либо выполнено, либо нет. Если при отправке возникнет исключение, метод должен откатиться.

```
1.  @Service
2.  public class ReportService {
3.      @Autowired
4.      private ReportRepository reportRepository;
5.
6.      @Transactional
7.      public void sendReport(long id) {
8.
9.          System.out.println(id + " sent");
10.         reportRepository.updatePublished(id);
11.
12.          //исключение заставит транзакцию откатиться:
13.          // throw new RuntimeException();
14.
15.     }
16. }
```

ITMO more than a UNIVERSITY

3.1. *REQUIRED* Propagation

REQUIRED is the default propagation. Spring checks if there is an active transaction, and if nothing exists, it creates a new one. Otherwise, the business logic appends to the currently active transaction:

```
@Transactional(propagation = Propagation.REQUIRED)
public void requiredExample(String user) {
    // ...
}
```

Furthermore, since *REQUIRED* is the default propagation, we can simplify the code by dropping it:

```
@Transactional
public void requiredExample(String user) {
    // ...
}
```

Let's see the pseudo-code of how transaction creation works for *REQUIRED* propagation:

```
if (isExistingTransaction()) {
    if (isValidateExistingTransaction()) {
        validateExisitingAndThrowExceptionIfNotValid();
    }
    return existing;
}
return createNewTransaction();
```

@Transactional

УНИВЕРСИТЕТ ИТМО

3.2. *SUPPORTS* Propagation

For *SUPPORTS*, Spring first checks if an active transaction exists. If a transaction exists, then the existing transaction will be used. If there isn't a transaction, it is executed non-transactional:

```
@Transactional(propagation = Propagation.SUPPORTS)
public void supportsExample(String user) {
    // ...
}
```

Let's see the transaction creation's pseudo-code for *SUPPORTS*:

```
if (isExistingTransaction()) {
    if (isValidateExistingTransaction()) {
        validateExisitingAndThrowExceptionIfNotValid();
    }
    return existing;
}
return emptyTransaction;
```

ITMO more than a UNIVERSITY

@Transactional

УНИВЕРСИТЕТ ИТМО

3.3. *MANDATORY* Propagation

When the propagation is *MANDATORY*, if there is an active transaction, then it will be used. If there isn't an active transaction, then Spring throws an exception:

```
@Transactional(propagation = Propagation.MANDATORY)
public void mandatoryExample(String user) {
    // ...
}
```

Let's again see the pseudo-code:

```
if (isExistingTransaction()) {
    if (isValidateExistingTransaction()) {
        validateExisitingAndThrowExceptionIfNotValid();
    }
    return existing;
}
throw IllegalTransactionStateException;
```

ITMO more than a UNIVERSITY

@Transactional

УНИВЕРСИТЕТ ИТМО

3.4. *NEVER* Propagation

For transactional logic with *NEVER* propagation, Spring throws an exception if there's an active transaction:

```
@Transactional(propagation = Propagation.NEVER)
public void neverExample(String user) {
    // ...
}
```

Let's see the pseudo-code of how transaction creation works for *NEVER* propagation:

```
if (isExistingTransaction()) {
    throw IllegalTransactionStateException;
}
return emptyTransaction;
```

@Transactional

УНИВЕРСИТЕТ ИТМО

3.5. *NOT_SUPPORTED* Propagation

If a current transaction exists, first Spring suspends it, and then the business logic is executed without a transaction:

```
@Transactional(propagation = Propagation.NOT_SUPPORTED)
public void notSupportedExample(String user) {
    // ...
}
```

The *JTATransactionManager* supports real transaction suspension out-of-the-box. Others simulate the suspension by holding a reference to the existing one and then clearing it from the thread context

ITMOre than a UNIVERSITY

3.6. *REQUIRES_NEW* Propagation

When the propagation is *REQUIRES_NEW*, Spring suspends the current transaction if it exists, and then creates a new one:

```
@Transactional(propagation = Propagation.REQUIRES_NEW)
public void requiresNewExample(String user) {
    // ...
}
```

Similar to *NOT_SUPPORTED*, we need the *JTATransactionManager* for actual transaction suspension.

The pseudo-code looks like so:

```
if (isExistingTransaction()) {
    suspend(existing);
    try {
        return createNewTransaction();
    } catch (exception) {
        resumeAfterBeginException();
        throw exception;
    }
}
return createNewTransaction();
```

3.7. *NESTED* Propagation

For *NESTED* propagation, Spring checks if a transaction exists, and if so, it marks a save point. This means that if our business logic execution throws an exception, then the transaction rollbacks to this save point. If there's no active transaction, it works like *REQUIRED*.

***DataSourceTransactionManager* supports this propagation out-of-the-box. Some implementations of *JTATransactionManager* may also support this.**

***JpaTransactionManager* supports *NESTED* only for JDBC connections. However, if we set the *nestedTransactionAllowed* flag to *true*, it also works for JDBC access code in JPA transactions if our JDBC driver supports save points.**

Finally, let's set the *propagation* to *NESTED*:

```
@Transactional(propagation = Propagation.NESTED)
public void nestedExample(String user) {
    // ...
}
```

@Propagation

УНИВЕРСИТЕТ ИТМО

Propagation отвечает на вопросы:

- что если два метода аннотированы *@Transactional,* и один вызывается из другого?
- Будет ли создано две транзакции, или же одна?
- Будет ли внутренний метод выбрасывать исключение, если снаружи нет никакой транзакции?

	Если вызывается из *@Transactional sendReport()*	**Если вызывается из *sendReport()* без *@Transactional*, либо вызывается отдельно.**
REQUIRED	используется существующая транзакция	транзакция создается
REQUIRED_NEW	создается отдельная вторая транзакция для внутреннего метода	транзакция создается
SUPPORTS	используется существующая транзакция	транзакция **не** создается
NOT_SUPPORTED	существующая транзакция **не** используется, код выполняется вне транзакции	транзакция **не** создается
NEVER	выбрасывает исключение	транзакция **не** создается
MANDATORY	используется существующая транзакция	выбрасывает исключение

- код выполняется вне транзакции (режим Auto-Commit)
- одна транзакция
- две транзакции
- исключение

@Isolation

УНИВЕРСИТЕТ ИТМО

4.2. *READ_UNCOMMITTED* Isolation

READ_UNCOMMITTED is the lowest isolation level and allows for the most concurrent access.

As a result, it suffers from all three mentioned concurrency side effects. A transaction with this isolation reads uncommitted data of other concurrent transactions. Also, both non-repeatable and phantom reads can happen. Thus we can get a different result on re-read of a row or re-execution of a range query.

We can set the *isolation* level for a method or class:

```
@Transactional(isolation = Isolation.READ_UNCOMMITTED)
public void log(String message) {
    // ...
}
```

Postgres does not support *READ_UNCOMMITTED* isolation and falls back to *READ_COMMITED* instead. Also, Oracle does not support or allow *READ_UNCOMMITTED*.

@Isolation

УНИВЕРСИТЕТ ИТМО

4.3. *READ_COMMITTED* Isolation

The second level of isolation, *READ_COMMITTED*, prevents dirty reads.

The rest of the concurrency side effects could still happen. So uncommitted changes in concurrent transactions have no impact on us, but if a transaction commits its changes, our result could change by re-querying.

Here we set the *isolation* level:

```
@Transactional(isolation = Isolation.READ_COMMITTED)
public void log(String message){
    // ...
}
```

***READ_COMMITTED* is the default level with Postgres, SQL Server, and Oracle.**

@Isolation

УНИВЕРСИТЕТ ИТМО

4.4. *REPEATABLE_READ* Isolation

The third level of isolation, *REPEATABLE_READ*, prevents dirty, and non-repeatable reads. So we are not affected by uncommitted changes in concurrent transactions.

Also, when we re-query for a row, we don't get a different result. However, in the re-execution of range-queries, we may get newly added or removed rows.

Moreover, it is the lowest required level to prevent the lost update. The lost update occurs when two or more concurrent transactions read and update the same row. *REPEATABLE_READ* does not allow simultaneous access to a row at all. Hence the lost update can't happen.

Here is how to set the *isolation* level for a method:

```
@Transactional(isolation = Isolation.REPEATABLE_READ)
public void log(String message){
    // ...
}
```

***REPEATABLE_READ* is the default level in Mysql. Oracle does not support *REPEATABLE_READ*.**

@Isolation

УНИВЕРСИТЕТ ИТМО

4.5. *SERIALIZABLE* Isolation

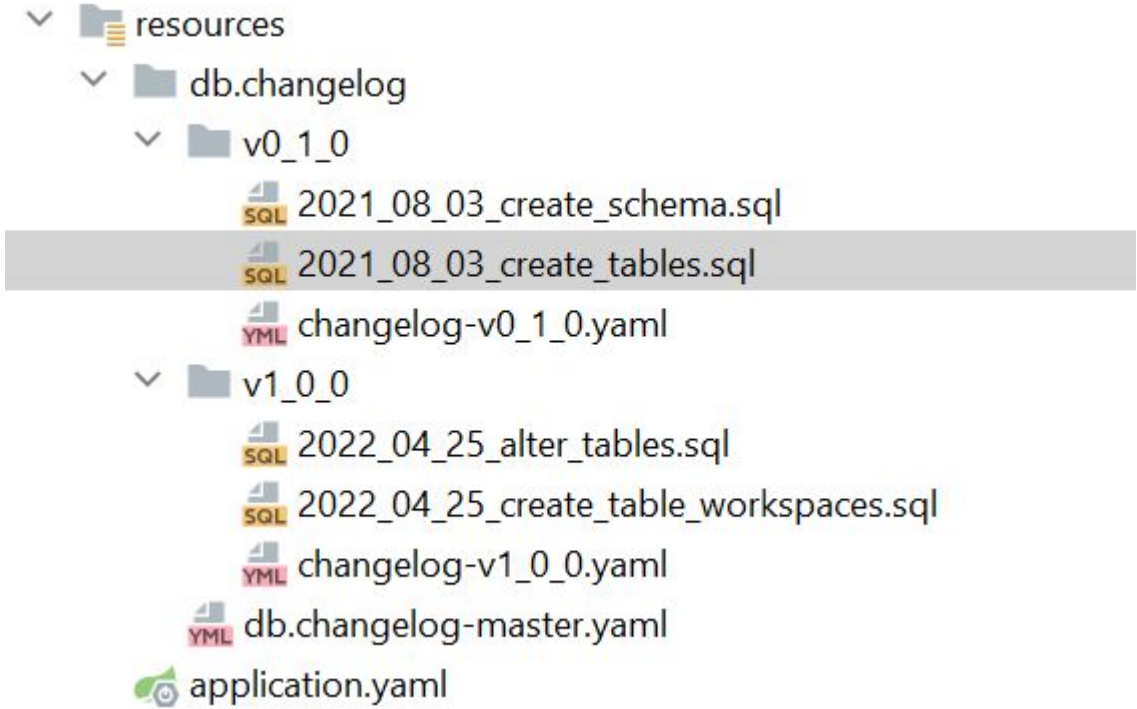
SERIALIZABLE is the highest level of isolation. It prevents all mentioned concurrency side effects, but can lead to the lowest concurrent access rate because it executes concurrent calls sequentially.

In other words, concurrent execution of a group of serializable transactions has the same result as executing them in serial.

Now let's see how to set *SERIALIZABLE* as the *isolation* level:

```
@Transactional(isolation = Isolation.SERIALIZABLE)
public void log(String message){
    // ...
}
```

Liquibase – sql файл



Liquibase – changelog-v0_1_0.yaml

УНИВЕРСИТЕТ ИТМО

```yaml
databaseChangeLog:
  - include:
      file: db/changelog/v0_1_0/2021_08_03_create_schema.sql
  - include:
      file: db/changelog/v0_1_0/2021_08_03_create_tables.sql
```

ITMO more than a UNIVERSITY

Liquibase – changelog-v1_0_0.yaml

УНИВЕРСИТЕТ ИТМО

```yaml
databaseChangeLog:
  - include:
      file: db/changelog/v1_0_0/2022_04_25_create_table_workspaces.sql
  - include:
      file: db/changelog/v1_0_0/2022_04_25_alter_tables.sql
```

ITMO more than a UNIVERSITY

Liquibase – db.changelog-master.yaml

УНИВЕРСИТЕТ ИТМО

```yaml
databaseChangeLog:
  - include:
      file: db/changelog/v0_1_0/changelog-v0_1_0.yaml
  - include:
      file: db/changelog/v1_0_0/changelog-v1_0_0.yaml
```

ITMO more than a UNIVERSITY

Liquibase – application.yaml

УНИВЕРСИТЕТ ИТМО

```yaml
spring:
  application:
    name: ${APPLICATION_NAME:NAME-service}
  liquibase:
    database-change-log-lock-table: ${MIGRATIONS_TABLE_LOCK:NAME_service_migrations_lock}
    database-change-log-table: ${MIGRATIONS_TABLE:NAME_service_migrations}
    liquibase-schema: ${MIGRATIONS_SCHEMA:migrations}
    url: jdbc:postgresql://${POSTGRES_HOST:localhost}:${POSTGRES_PORT:5432}/${POSTGRES_DB:postgres}
    user: ${POSTGRES_USERNAME:postgres}
    password: ${POSTGRES_PASSWORD:postgres}
```

Testcontainers

УНИВЕРСИТЕТ ИТМО

```
@Testcontainers
@SpringBootTest
public class AbstractIntegrationBaseTest {
    private final static PostgreSQLContainer<?> POSTGRES_CONTAINER;

    // singleton aka global container
    static {
        POSTGRES_CONTAINER =
                new PostgreSQLContainer<>( dockerImageName: "postgres:13")
                        .withDatabaseName("postgres")
                        .withUsername("postgres")
                        .withPassword("postgres");
        POSTGRES_CONTAINER.start();
    }

    @DynamicPropertySource
    static void init(DynamicPropertyRegistry registry) {
        var urlPg : String = String.format(
                "%s:%d/%s",
                POSTGRES_CONTAINER.getHost(),
                POSTGRES_CONTAINER.getFirstMappedPort(),
                "postgres"
        );
        registry.add( name: "spring.datasource.url", () -> "jdbc:postgresql://" + urlPg + "?schema=NAME_service");
        registry.add( name: "spring.liquibase.url", () -> "jdbc:postgresql://" + urlPg);
        // needed because default is migrations
        registry.add( name: "spring.liquibase.liquibase-schema", () -> "public");
    }
}
```

Testcontainers – base test

УНИВЕРСИТЕТ ИТМО

```
21  @Slf4j
22  class GeometryControllerTest extends AbstractIntegrationBaseTest {
23      @Autowired
24      private GeometryController geometryController;
25      @Autowired
26      private WorkspaceController workspaceController;
27
28      @AfterEach
29      void tearDown() {
30          var workspaces : List<WorkspaceDto> = workspaceController.findAll(WorkspaceFilterDto.builder().build())
31                  .collectList()
32                  .share().block();
33
34          if (workspaces != null) {
35              for (var workspace : workspaces) {
36                  workspaceController.delete(workspace.getId()).share().block();
37              }
38          }
39      }
```

ITMO more than a UNIVERSITY

Testcontainers – junit-jupiter

УНИВЕРСИТЕТ ИТМО

```
29  @Test
30  fun assignAssignedArtistError() {
31      val error = assertThrows<IllegalStateException> {
32          artistAction.assignArtist(zvukId = 1, user = user)
33          artistAction.assignArtist(zvukId = 1, user = user)
34      }
35      assertEquals(messageTranslationService.byCode( code: "ARTIST_ALREADY_ASSIGNED"), error.message)
36  }
37
38  @Test
39  fun myArtistsTest() {
40      val zvukIds = listOf(1L, 2L, 3L)
41      for (zvukId in zvukIds) {
42          artistAction.assignArtist(zvukId = zvukId, user = user)
43      }
44
45      val artists = artistAction.myArtists(user = user)
46      val artistsPage1 = artistAction.myArtists(size = 2, page = 0, user = user)
47      val artistsPage2 = artistAction.myArtists(size = 2, page = 1, user = user)
48
49      assertEquals(artists.items?.filter { artist -> zvukIds.contains(artist.zvukId) }?.size,  actual: 3)
50
51      assertEquals(artists.items?.size,  actual: 3)
52      assertEquals(artistsPage1.items?.size,  actual: 2)
53      assertEquals(artistsPage2.items?.size,  actual: 1)
54  }
```

ITMOre than a UNIVERSITY

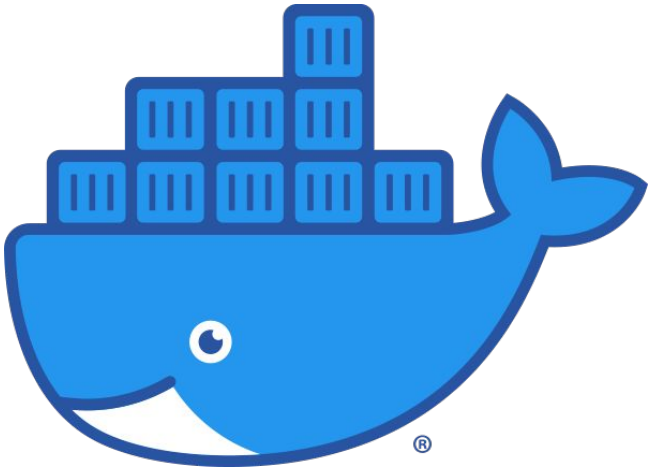
junit-Jupiter – нет миграций

УНИВЕРСИТЕТ ИТМО

```
76  @Test
77  @SqlGroup(
78    Sql(value = ["/sql/init_handbooks.sql"], executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD),
79    Sql(value = ["/sql/registration/init.sql"], executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD),
80    Sql(value = ["/sql/representative/init.sql"], executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD),
81    Sql(value = ["/sql/user/init.sql"], executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD),
82    Sql(value = ["/sql/truncate_models.sql"], executionPhase = Sql.ExecutionPhase.AFTER_TEST_METHOD),
83    Sql(value = ["/sql/registration/truncate.sql"], executionPhase = Sql.ExecutionPhase.AFTER_TEST_METHOD),
84    Sql(value = ["/sql/representative/truncate.sql"], executionPhase = Sql.ExecutionPhase.AFTER_TEST_METHOD)
85  )
86  fun registrationInvalidCodeTest() {
87    val error = assertThrows<IllegalStateException> {
88      modelContext.refreshContext()
89
90      registrationAction.generateCode(phone)
91      registrationAction.confirmCode(phone, "1111")
92    }
93    assertEquals("MESSAGE_INVALID_CODE_ERROR", error.message)
94  }
```

ITMO more than a UNIVERSITY

Postgres in docker

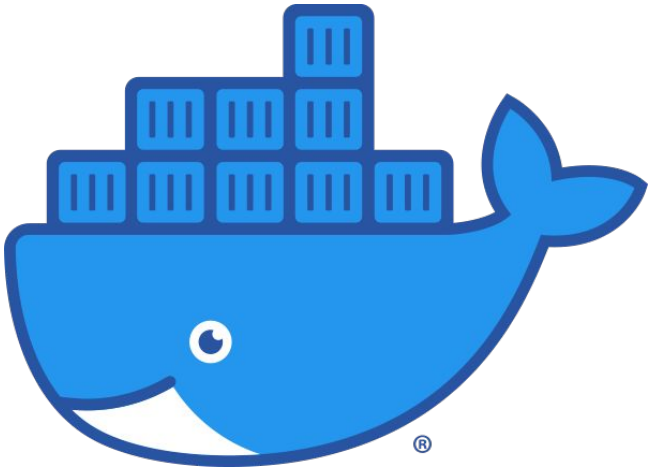


docker-compose.yaml

C: > Users > andermirik > Desktop > pg > docker-compose.yaml

```yaml
version: "3.7"

services:
  aurora_postgres:
    image: postgres:14
    container_name: aurora_postgres
    restart: on-failure
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_DB=postgres
    volumes:
      - ./db-data:/var/lib/postgresql/data
      - ./public.sql:/docker-entrypoint-initdb.d/create_tables.sql

```

Dockerfile



```
FROM gradle:6.6.1-jdk11 AS build
COPY --chown=gradle:gradle . /home/gradle/src
WORKDIR /home/gradle/src
RUN --mount=type=cache,id=gradle,target=/home/gradle/.gradle  gradle build --no-daemon -x test
################################################################################
FROM ubuntu:18.04

RUN apt-get -y update &&\
    apt-get -y upgrade &&\
    apt-get install -y git &&\
    apt-get install -y ffmpeg &&\
    apt-get install -y default-jre &&\
    apt-get install -y openjdk-11-jdk

RUN mkdir /app

RUN mkdir /yolo

COPY ./best.torchscript /yolo/best.torchscript
COPY ./face_mask.yaml /yolo/face_mask.yaml

COPY ./lib /opencvlib

COPY --from=build /home/gradle/src/build/libs/*.jar \
      /app/spring-boot-application.jar
#
#
#
ENTRYPOINT [\
  "java",\
  "-XX:+UseContainerSupport",\
  "-XX:-UseAdaptiveSizePolicy",\
  "-Xmx128m",\
  "-Xms32m",\
  "-Xss256k",\
  "-Djava.library.path=/opencvlib/",\
  "-Djava.security.egd=file:/dev/./urandom",\
  "-jar",\
  "/app/spring-boot-application.jar"]

```

Спасибо за внимание!

www.ifmo.ru

ITMOre than a UNIVERSITY