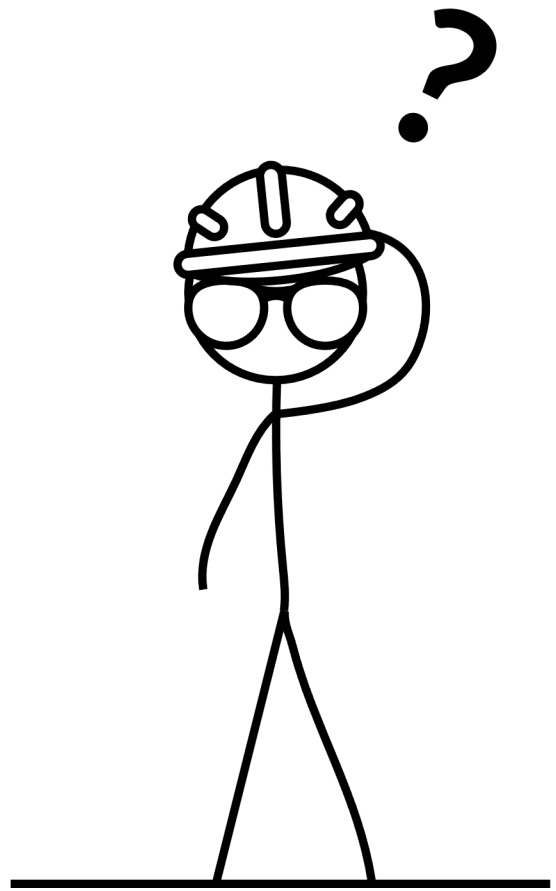
HAULMONT

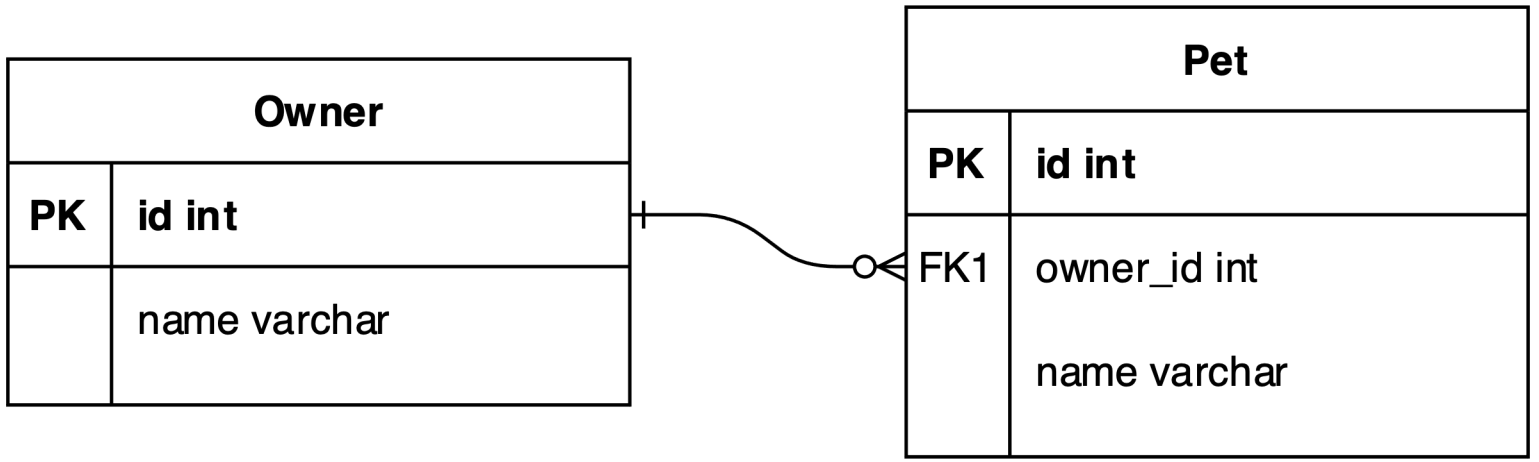
JPA Puzzlers

+Hibernate и Spring



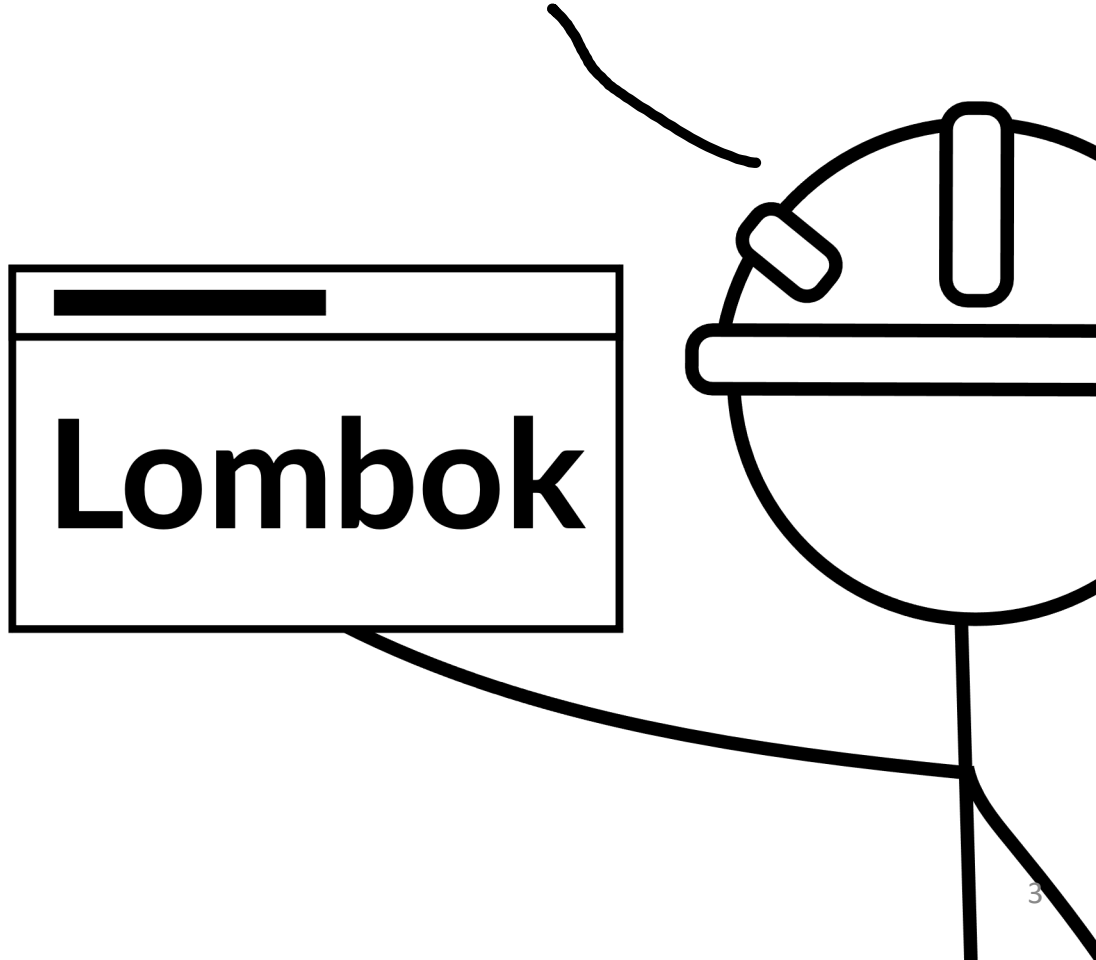
Андрей Беляев @belyaev_andrey

Схема данных



JPA и ...

Что может пойти не так?



JPA Entities

```java
@Entity
public class Owner {

    @Id
    private Long id;

    private String name;

}
```

- Аннотация @Entity
- Аннотация @Id
- Класс верхнего уровня
- Не финальный
- Конструктор по умолчанию
- Getters/Setters

- Lombok помогает убрать boilerplate

Сущность с Lombok

```
@Data
@Entity
public class Owner {

    @Id
    private Long id;

    private String name;

}
```

- @ToString
- @EqualsAndHashCode
- @Getter
- @Setter
- @RequiredArgsConstructor

Исходные данные

```java
@Data
@Table(name = "owner")
@Entity
public class Owner {
    @Id
    @Column(name = "id", nullable = false)
    private Long id;

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

    @OneToMany(mappedBy = "owner")
    private Set<Pet> pets;
}
```

```java
@Data
@Table(name = "pet")
@Entity
public class Pet {
    @Id
    @Column(name = "id", nullable = false)
    private Long id;

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

    @ManyToOne
    @JoinColumn(name = "owner_id")
    private Owner owner;
}
```

Что произойдет?

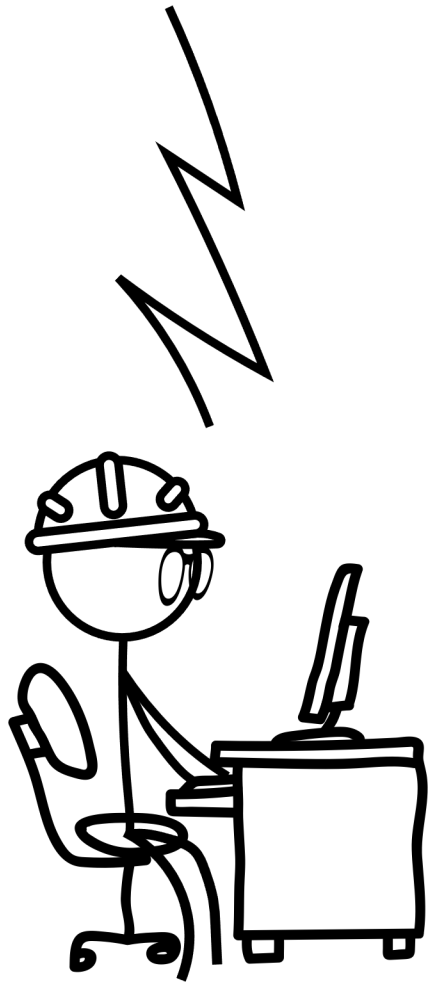
```
ownerRepository.findById(1L).ifPresent(System.out::println);
```

1. Owner(id=1, name=Jane, pets=[Pet(id=1, name=Fido)])
2. NullPointerException
3. LazyInitException
4. Owner@1234
5. Owner(id=1, name=Jane)

id	name
1	Jane
2	John

id	name	owner_id
1	Fido	1
2	Mimi	2

Coding Time



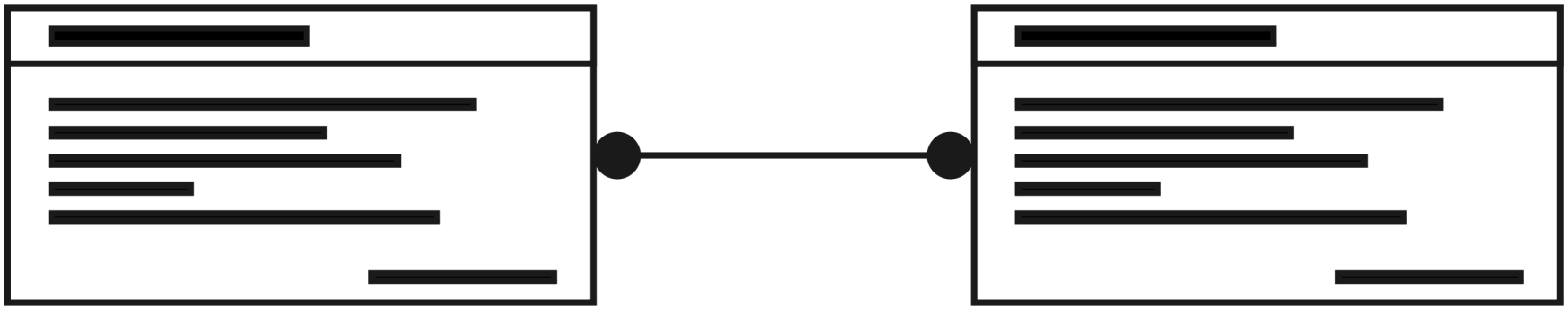
Почему так происходит?

- Lombok не читает мысли
- И не понимает, что это JPA сущность
- Общего подхода для equals/hashcode нет

Итого

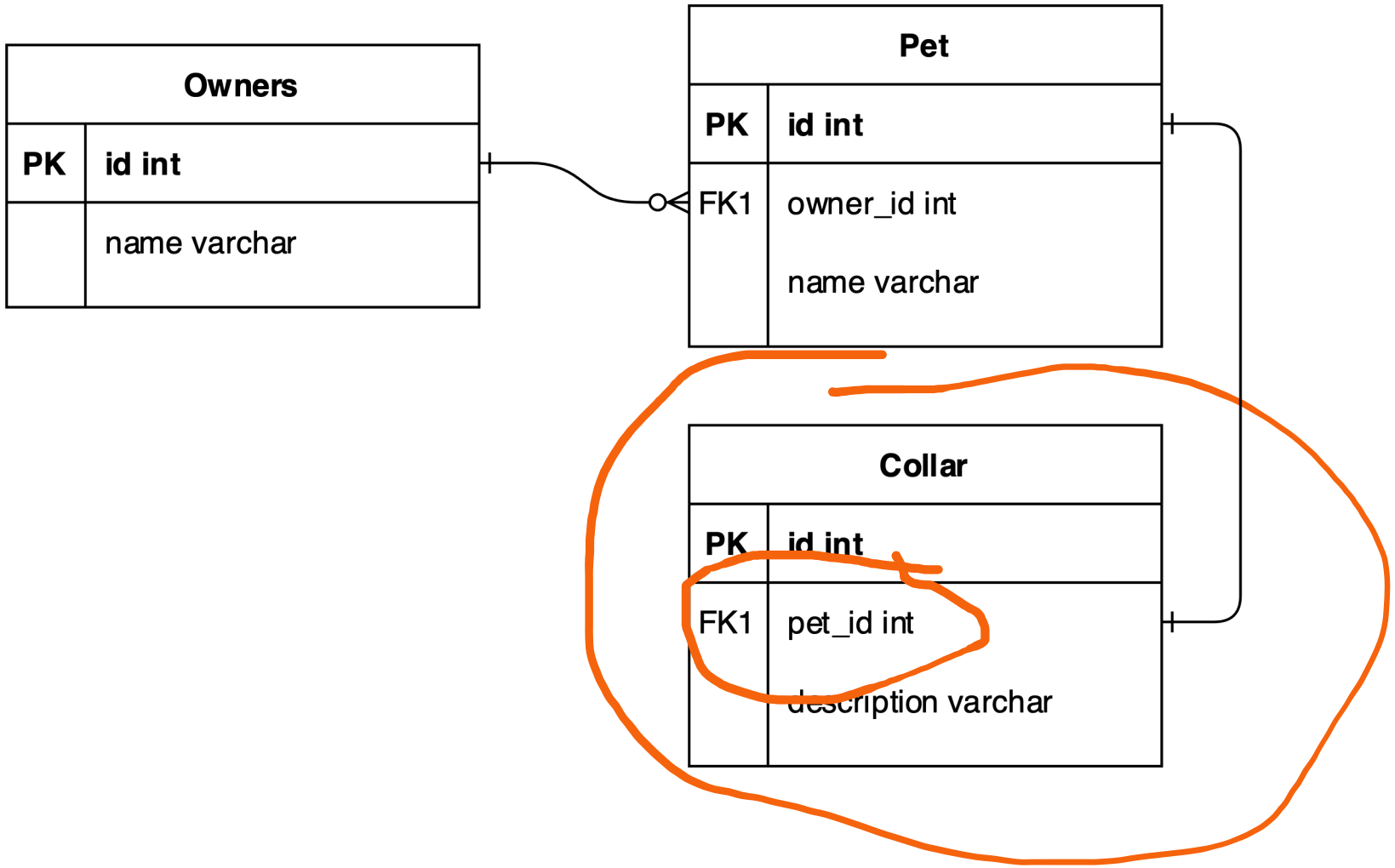
- Сгенерированный код – не ваш
- Аккуратно избавляйтесь от boilerplate
  - Get/Set
  - NoArgsConstructor
- Помните про то, что сущности – мутабельны
  - equals/hashCode
  - toString

*ToOne отношения



Когда LAZY не помогает

Исходные данные



Особенности *ToOne

- По умолчанию – EAGER
  - +1 запрос
  - +1 join
- А если сделаем LAZY?

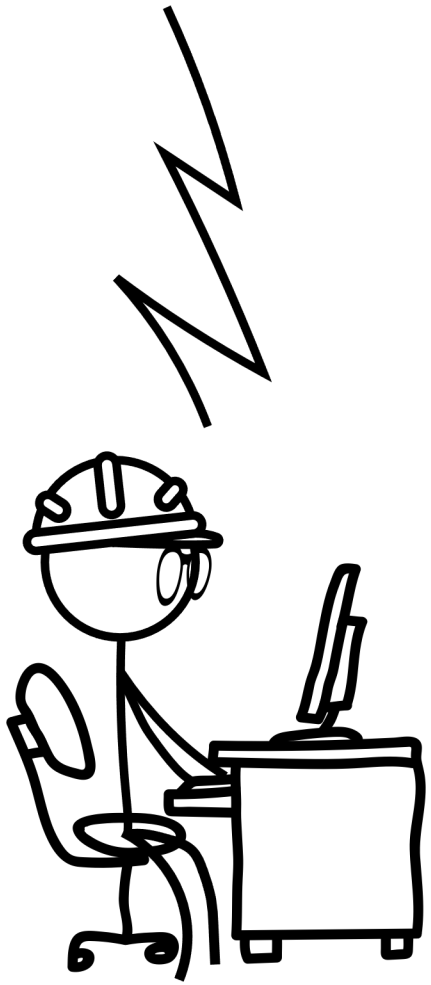
```java
@Entity
public class Pet {

    @Id
    @Column(name = "id", nullable = false)
    private Long id;

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

    @OneToOne(mappedBy = "pet", fetch = FetchType.LAZY)
    private Collar collar;
```

Что произойдет?

```
@OneToOne(mappedBy = "pet", fetch = FetchType.LAZY)
private Collar collar;
```

1. Поможет
2. Не поможет
3. Поможет, если поставить optional=false

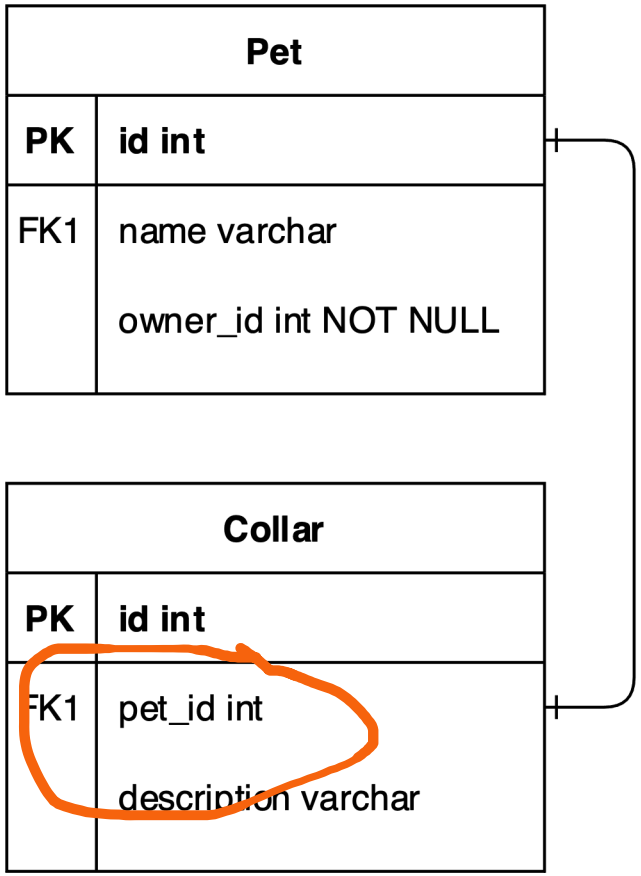
Coding Time



Почему так происходит?

- Чтобы загрузить Collar, нужно знать его ID
- Иначе hibernate не сможет загрузить объект
- Optional связь – отдельный случай

```
The parent-side @OneToOne association requires
bytecode enhancement so that the association can
be loaded lazily.

Otherwise, the parent-side association is always
fetched even if the association is marked
with FetchType.LAZY.
```



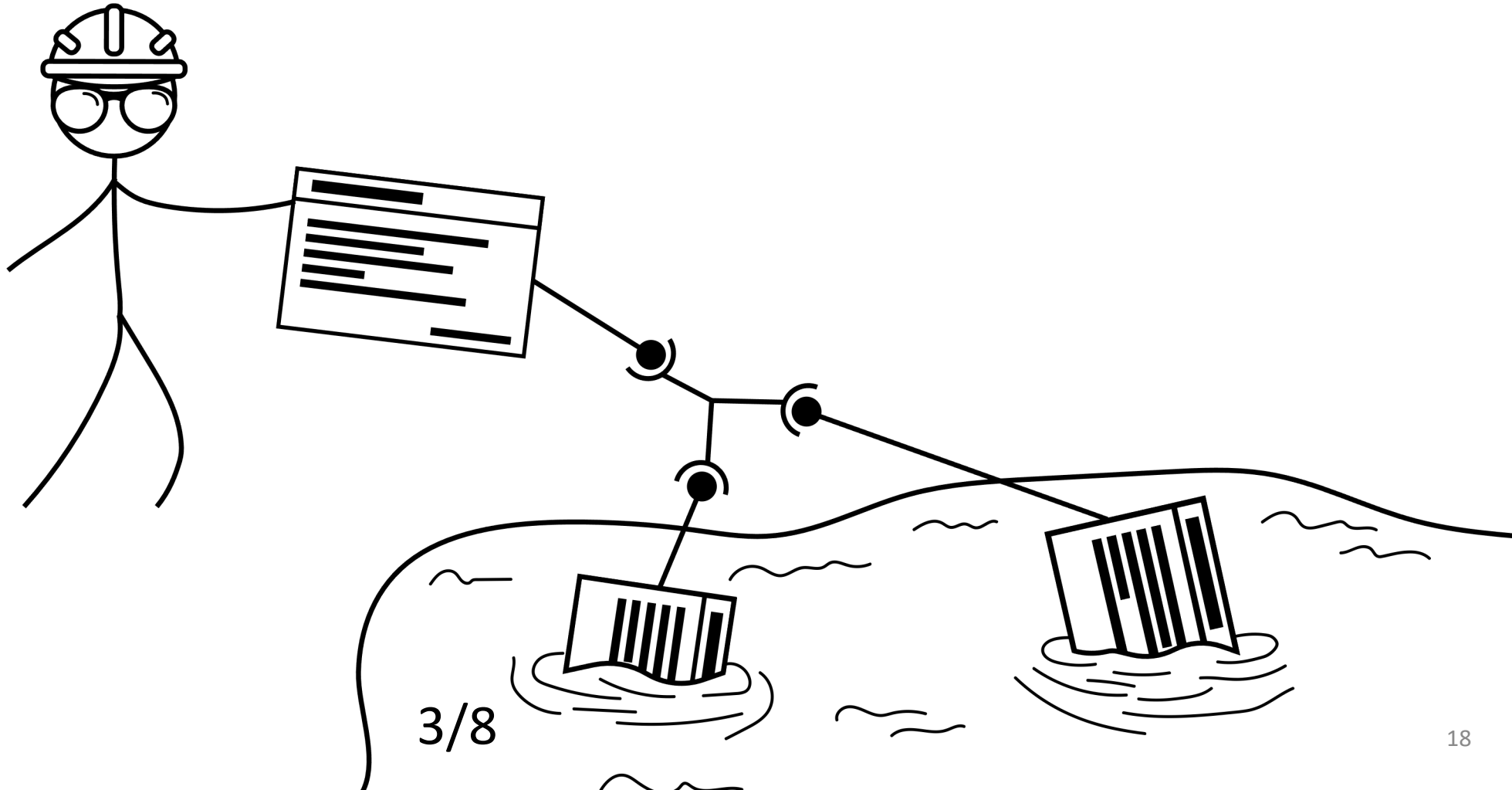
Итого

- Выставление FetchType не решает проблемы автоматом
- У ORM могут быть свои соображения
- Внимательно следите за Owning Side
- Можно использовать Bytecode Enhancer
- Или @MapsId

https://vladmihalcea.com/the-best-way-to-map-a-onetoone-relationship-with-jpa-and-hibernate/

Графы сущностей

a.k.a. Entity Graphs

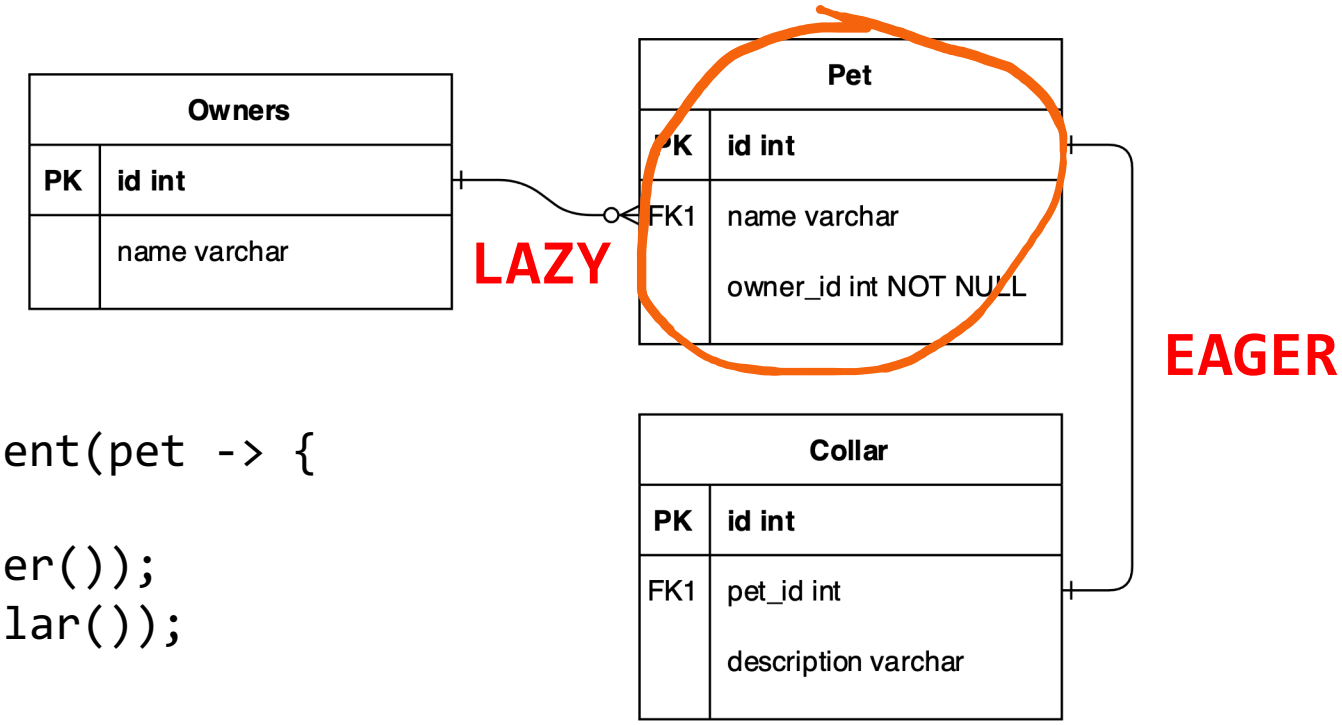


Графы сущностей

- Часть спецификации JPA
- Способ выборки связанных сущности
- Boot-time валидация

Исходные данные

```
@NamedEntityGraphs({
        @NamedEntityGraph(
                name = "pet-with-owner",
                attributeNodes = {@NamedAttributeNode("owner")}
        )
})
```

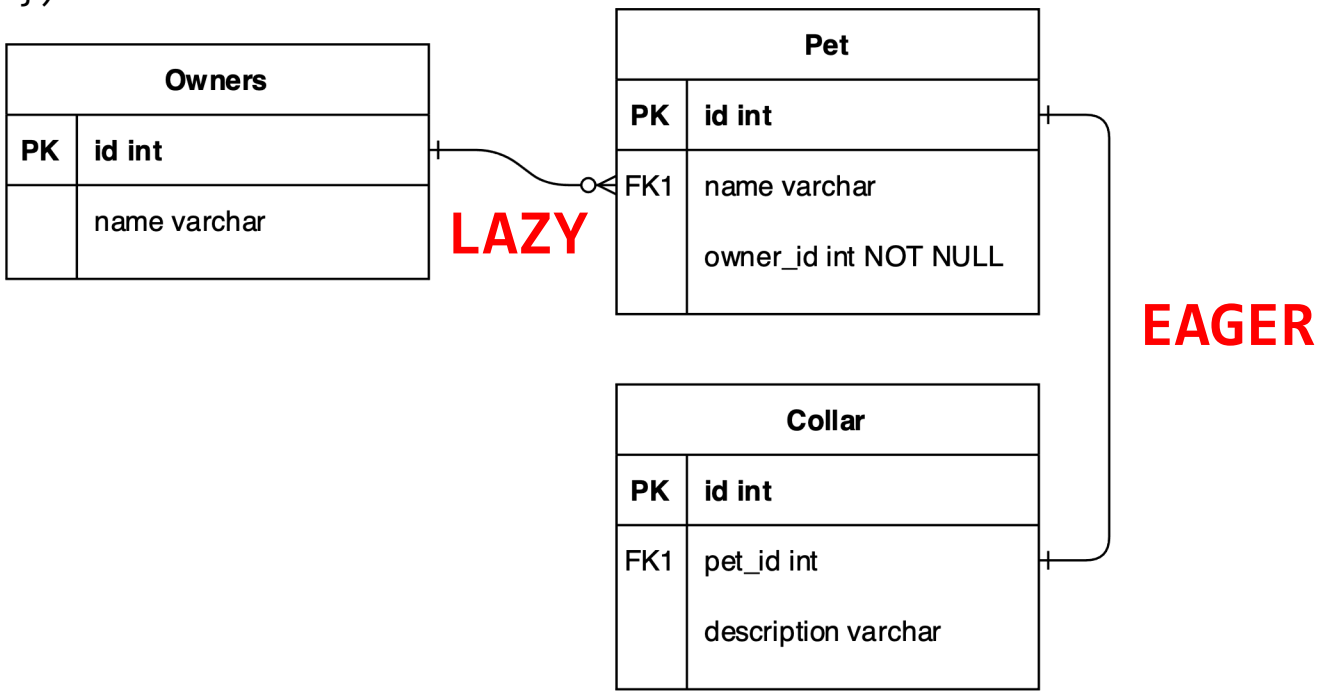



```
petRepository.findById(1L).ifPresent(pet -> {
    System.out.println(pet);
    System.out.println(pet.getOwner());
    System.out.println(pet.getCollar());
});
```

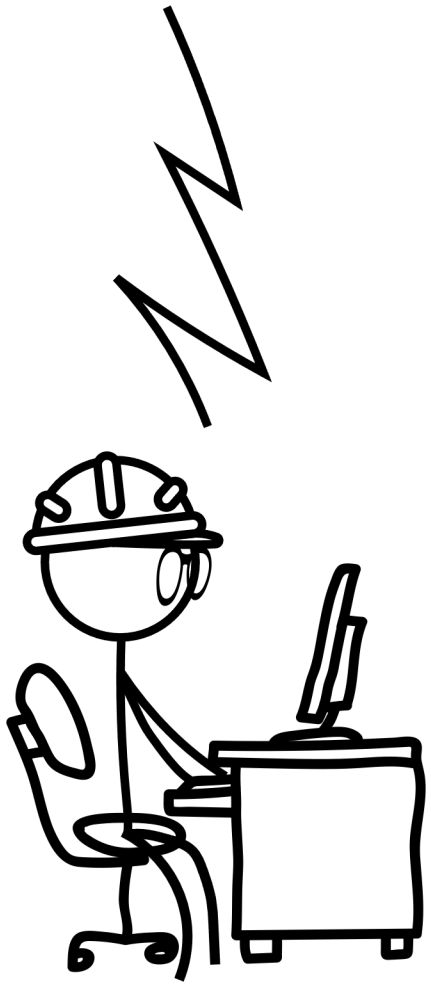
Что произойдет?

1. Все распечатается
2. Будет +1 запрос к Collar
3. LazyInitException
4. Boot-time error

```
@NamedEntityGraphs({
        @NamedEntityGraph(
                name = "pet-with-owner",
                attributeNodes = {@NamedAttributeNode("owner")}
        )
})
```



Coding Time



Почему так происходит?

- Все строго по спецификации
- Два режима работы Entity Graph
  - Fetch – выборка только графа
  - Load – граф + EAGER

Итого

- Можно использовать Entity Graph вместо LAZY/EAGER
  - Гибкая настройка
  - Нормальный SQL запрос
  - Применяется не только для findAll
- Выборка локальных атрибутов зависит от провайдера
- Но надо помнить про FETCH/LOAD

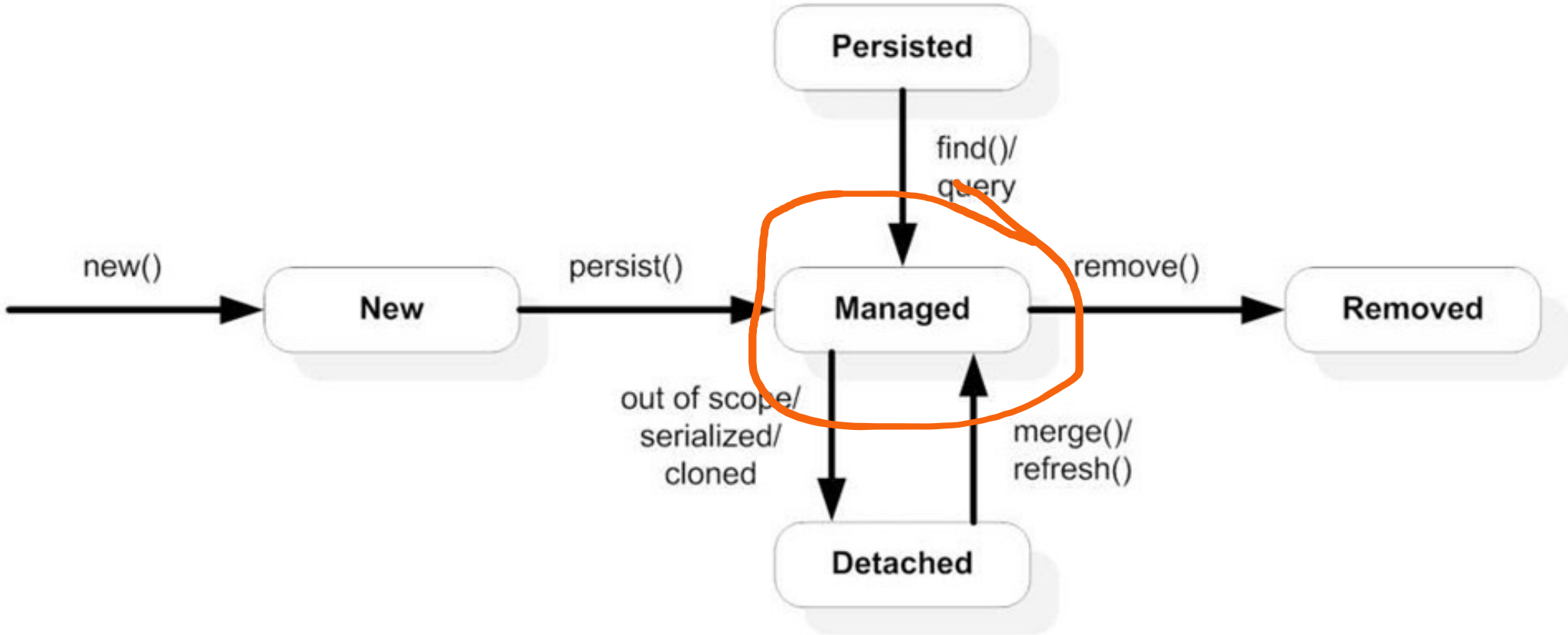
```
The persistence provider is permitted to fetch additional entity
  state beyond that specified by a fetch graph or load graph.
```

Кэширование

Помощь или помеха?

4/8

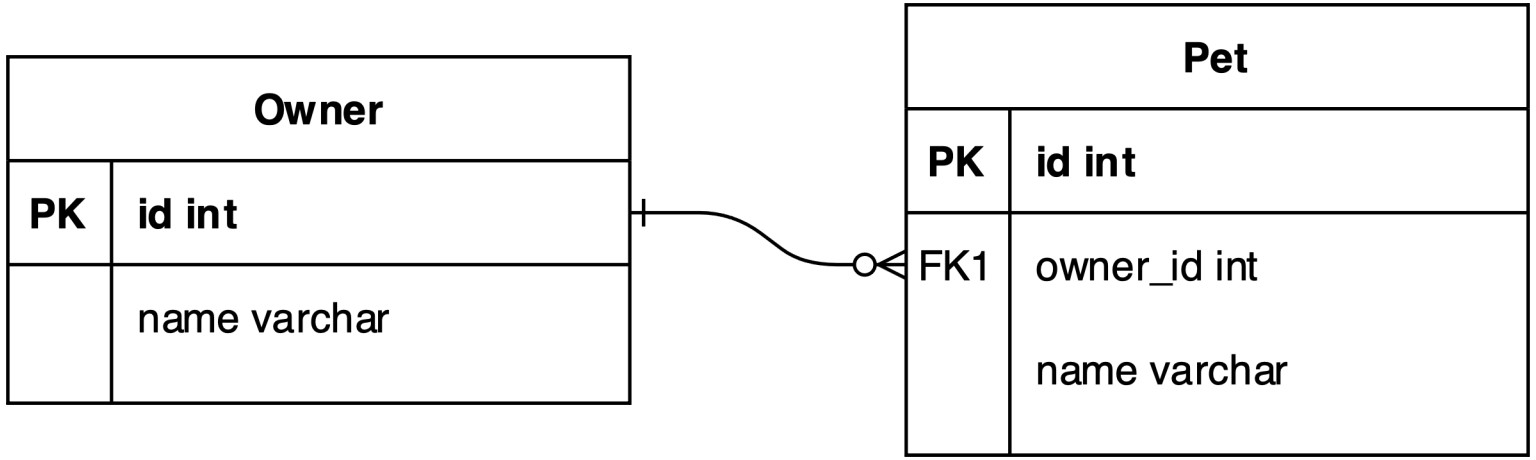
Состояния сущности



L1 Cache

- Сохраняет managed сущности на время транзакции
  - Класс PersistenceContext в Hibernate
- Один на поток выполнения
- Поддерживается автоматически

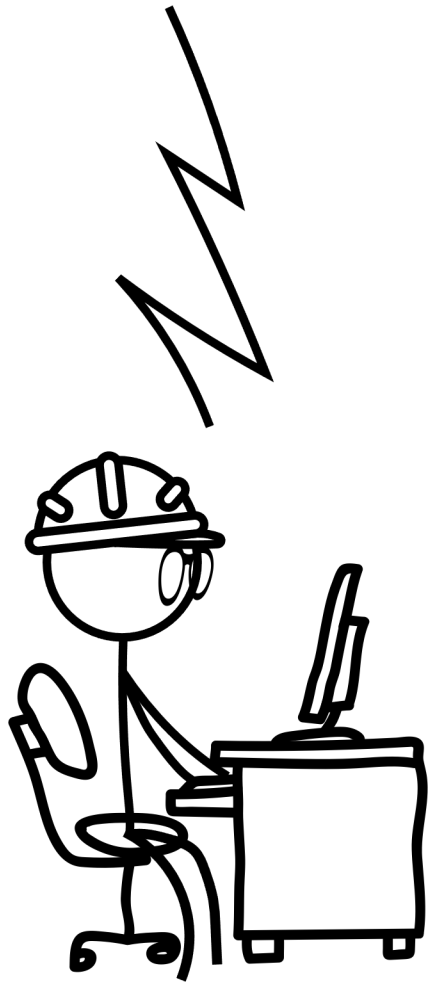
Исходные данные




```
Owner owner = ownerService.getOwnerOnly(1L);
System.out.println(owner);
```

```
Owner(id = 1, name = Jane)
```

Исходные данные

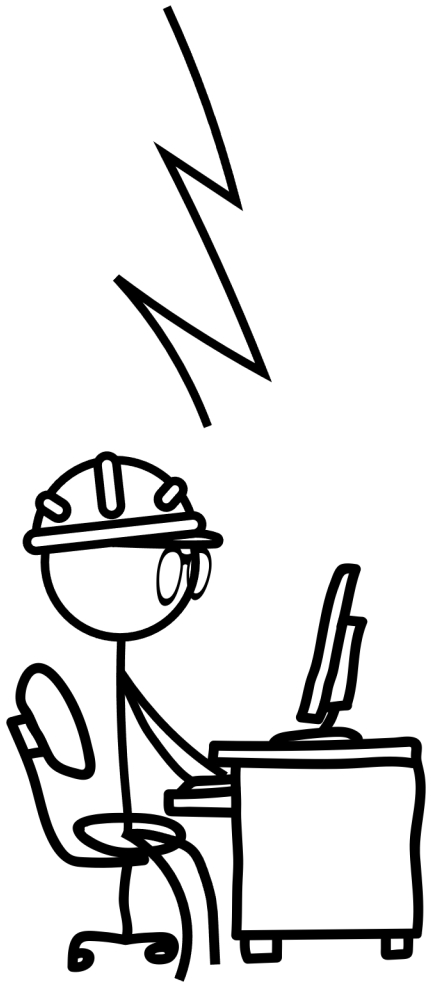


Что произойдет?

```
Owner owner = ownerService.getOwnerOnly(1L);
System.out.println(owner);
owner.getPets().forEach(System.out::println);
```

- LazyInitException
- Питомцы распечатаются нормально
- NullPointerException
- JpqlTransactionExecutionException

Coding Time



Почему так происходит?

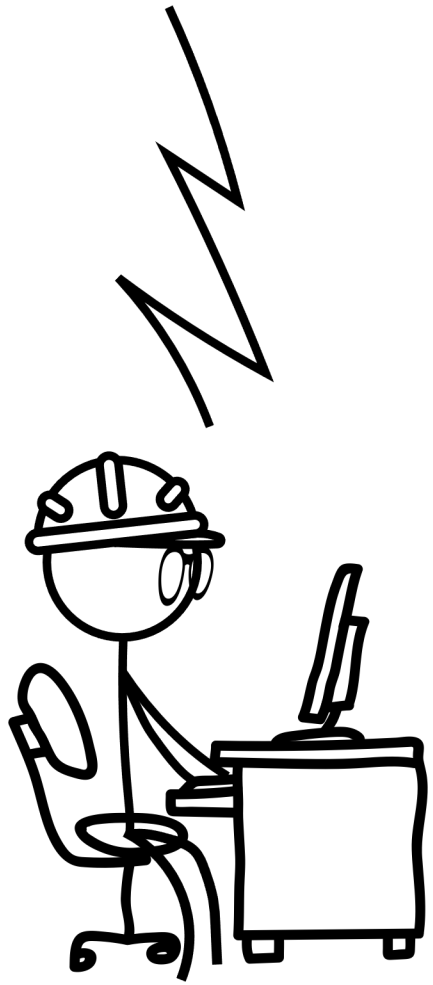
- Все происходит в одной транзакции
- Один L1 кэш
- Экземпляры объектов JPA одни и те же!
  - Подгружаем lazy атрибуты
  - И получаем то, что получаем

Кто за это отвечает?

DefaultLoadEventListener

```
private Object doLoad(EntityKey keyToLoad) {

   PersistenceContextEntry persistenceContextEntry =
                  CacheEntityLoaderHelper.INSTANCE.loadFromSessionCache(keyToLoad);

   Object entity = persistenceContextEntry.getEntity();

   if ( entity != null ) {
      return persistenceContextEntry.isManaged() ? entity : null;
   }

   entity = CacheEntityLoaderHelper.INSTANCE.loadFromSecondLevelCache( event, persister, keyToLoad );

   if ( entity == null ) {
       entity = loadFromDatasource( event, persister );
   }

   return entity;
}
```

Coding Time



Что произойдет?

- Owner(id = 1, name = Jill)
- Owner(id = 1, name = Jane)
- IllegalEntityStateException
- LazyInitException

Почему так происходит?

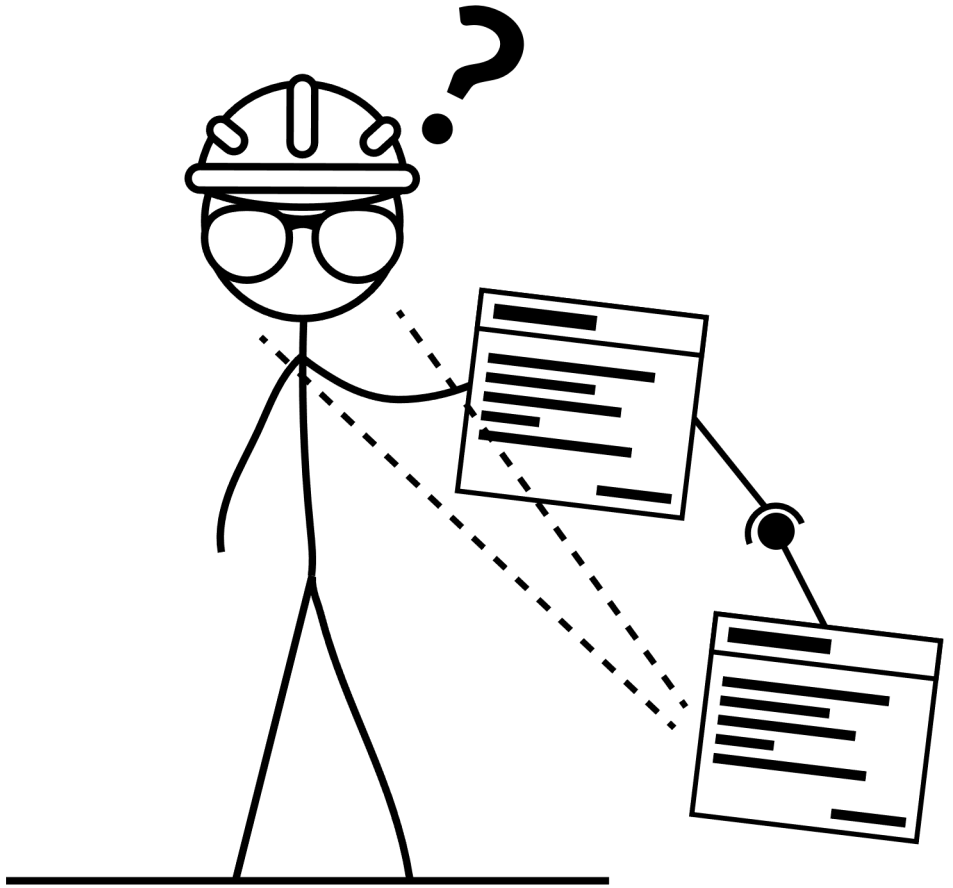
- Изменения проходят мимо L1 кэша
- Невыгодно выбирать изменяемые сущности

Итого

- Объекты внутри транзакции – одни и те же!
  - Но не всегда
- Следите за состоянием объектов
- Разделяйте действия по транзакциям
- При изменениях иногда нужно сбрасывать кэш

Неожиданная выборка

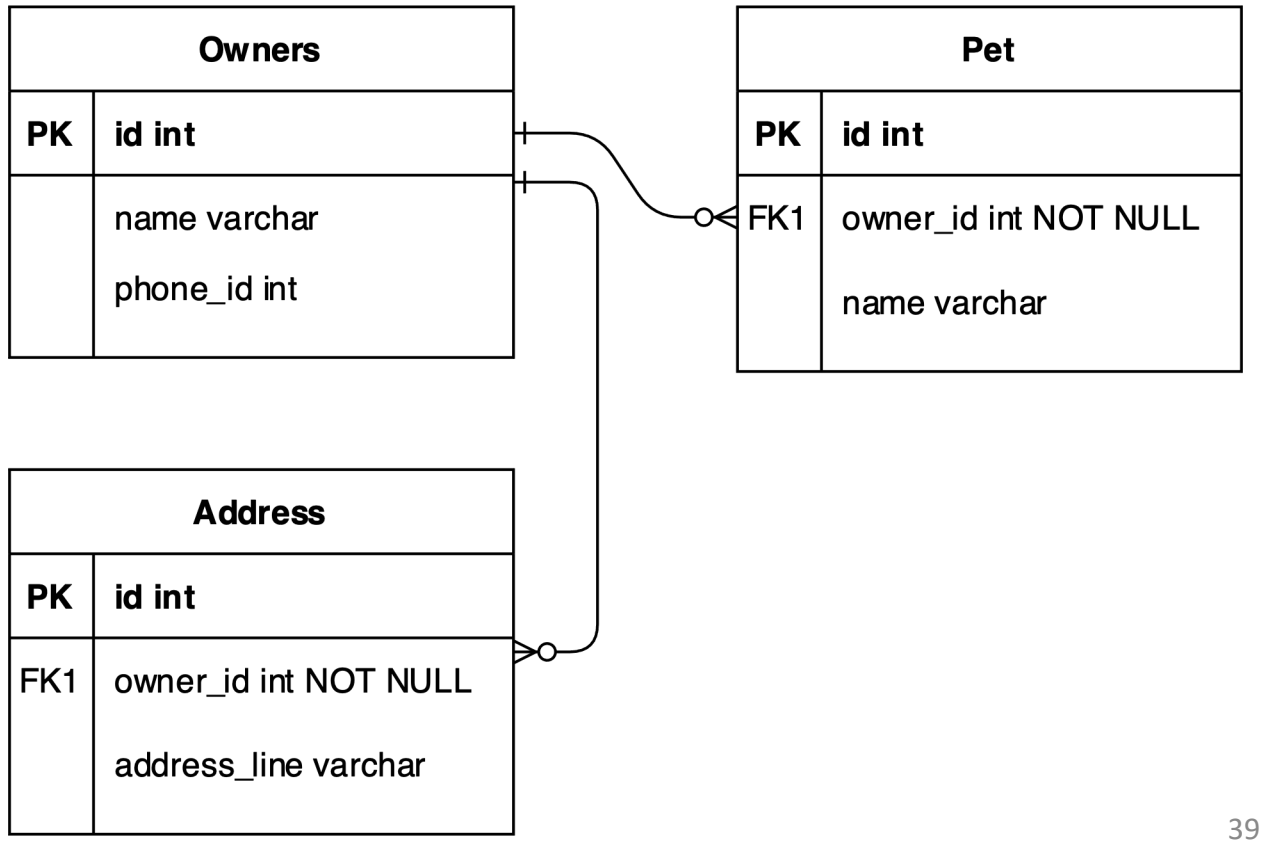
Хотя все было LAZY



5/8

Каскадные операции

- Удобный способ избежать возни с master-detail
- Но есть несколько «но»



Исходные данные

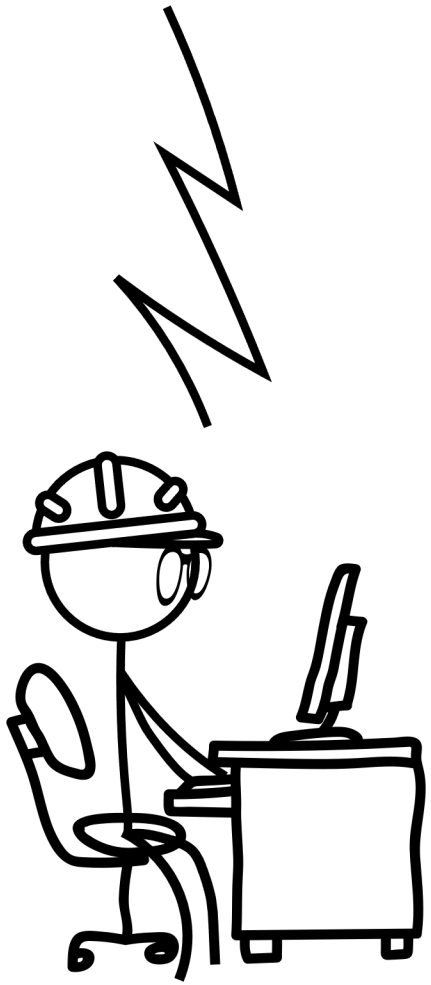
```
@Table(name = "owner")
@Entity
public class Owner {
    @Id
    @Column(name = "id", nullable = false)
    private Long id;

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

    @OneToMany(mappedBy = "owner", cascade = CascadeType.MERGE)
    private Set<Pet> pets;

    @OneToMany(mappedBy = "owner", cascade = CascadeType.MERGE)
    private Set<Address> addresses;
}
```

Что произойдет?

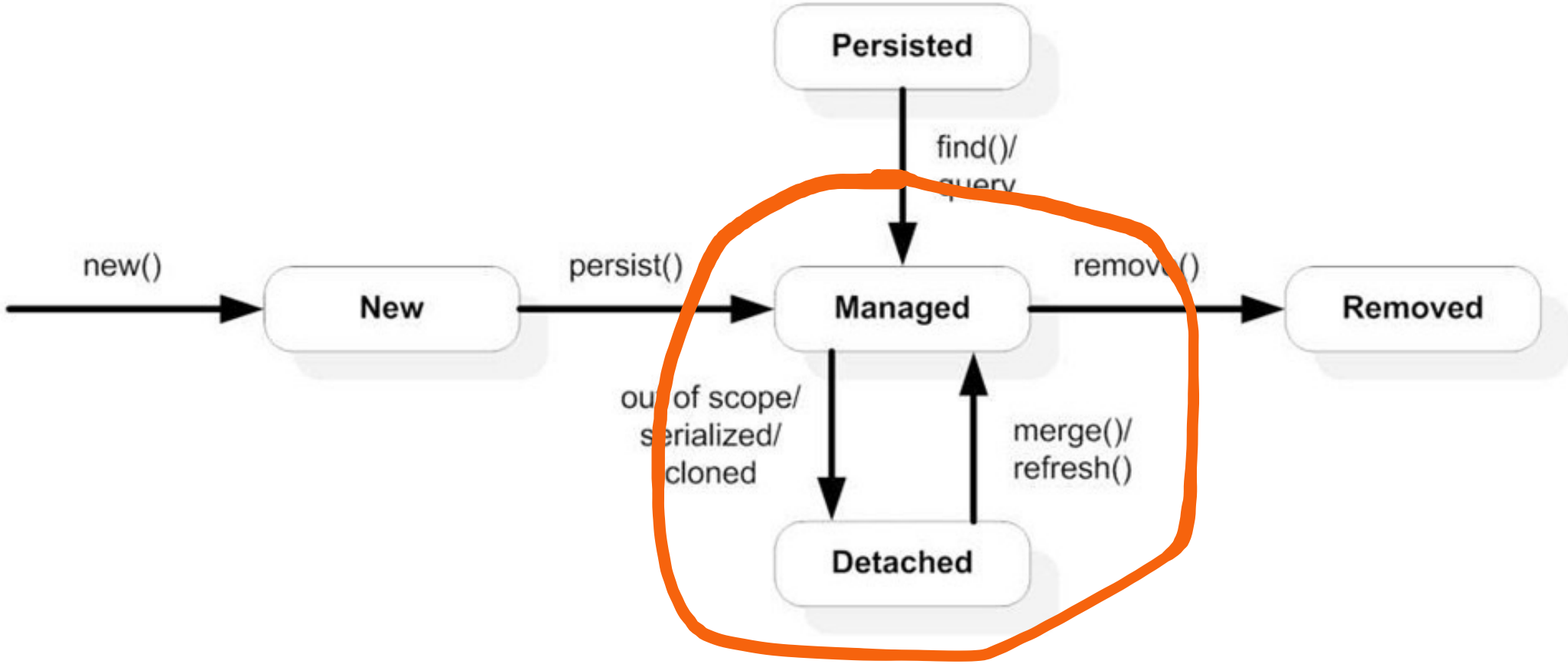
```
public void run(String... args) throws Exception {
    Owner owner = ownerRepository.findById(1L).orElseThrow();
    owner.setName("Johnny Doe "+System.nanoTime());
    ownerRepository.save(owner);
}
```

- 1 select, 1 update
- 2 select, 1 update
- 3 select, 1 update
- LazyInitException

Coding Time

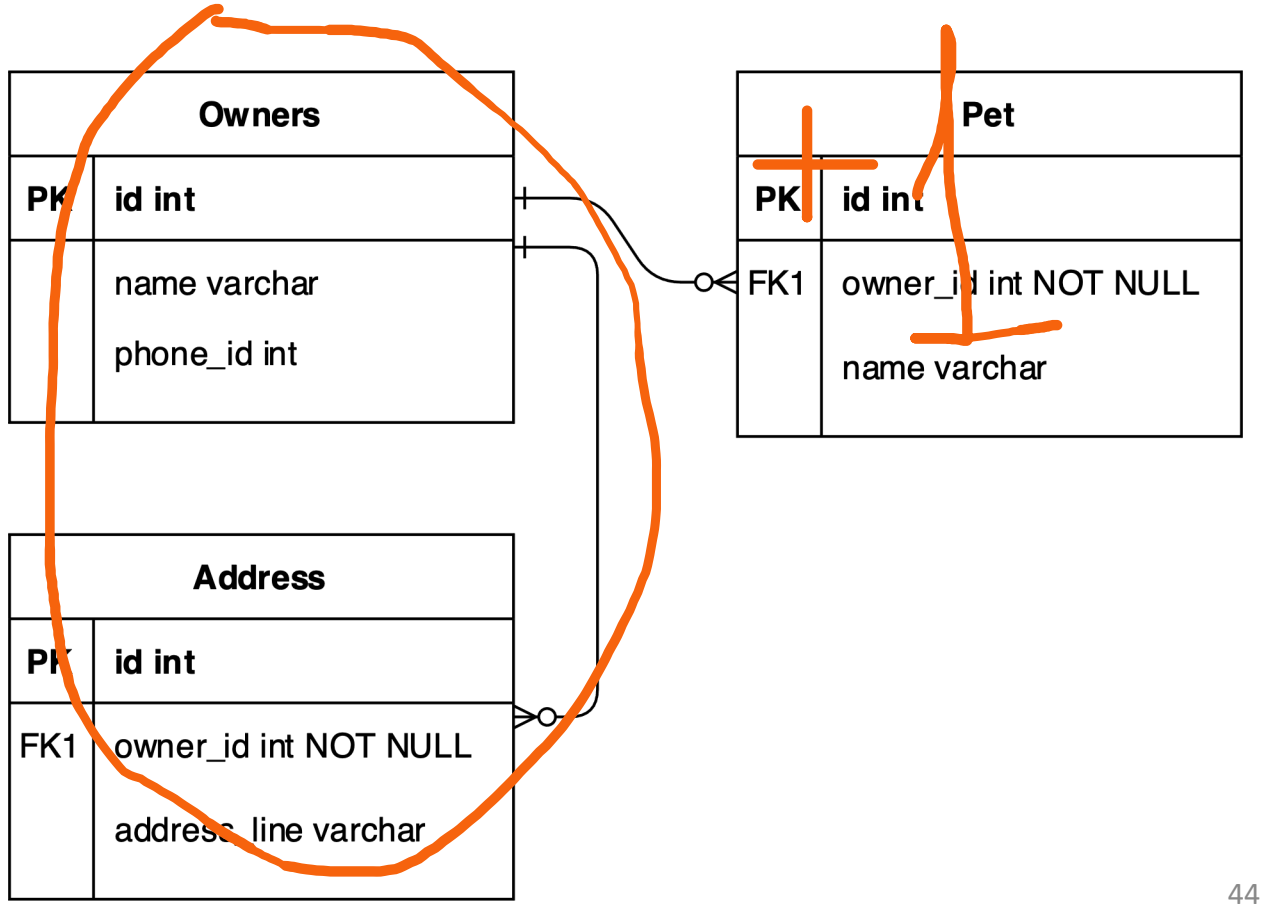


Почему так происходит?

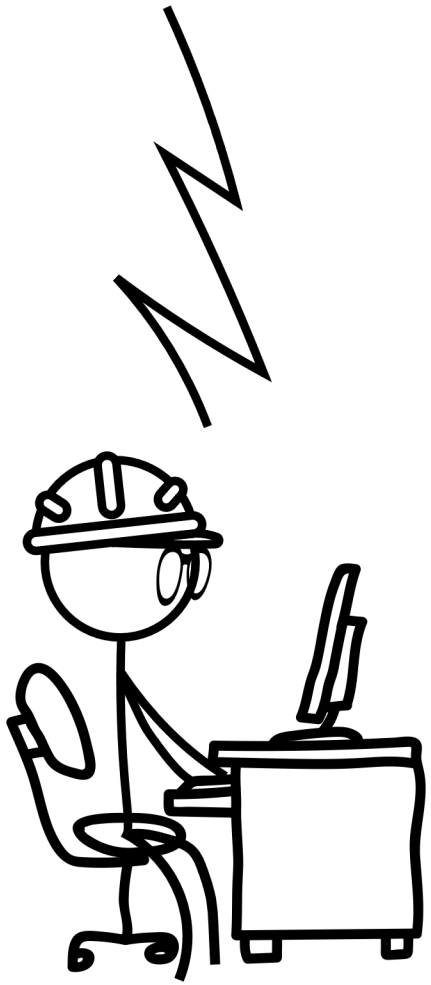


Почему так происходит?

- MERGE выполняет выборку из базы
- Обновляется граф сущности
- Но не весь!



Coding Time



Кто за это отвечает?

JoinWalker

```
private void addAssociationToJoinTreeIfNecessary() {
  if ( joinType != JoinType.NONE ) {
    addAssociationToJoinTree(
        type,
        aliasedLhsColumns,
        alias,
        path,
        currentDepth,
        joinType
    );
  }
}
```

```
protected JoinType getJoinType() {
…
if ( isTooDeep( currentDepth ) ||
            ( associationType.isCollectionType() &&
isTooManyCollections() ) ) {
    return JoinType.NONE;
  }
…
return getJoinType( nullable, currentDepth );
}
```

Кто за это отвечает?

CascadeEntityJoinWalker

```
@Override
protected boolean isTooManyCollections() {
  return countCollectionPersisters( associations ) > 0;
}
```

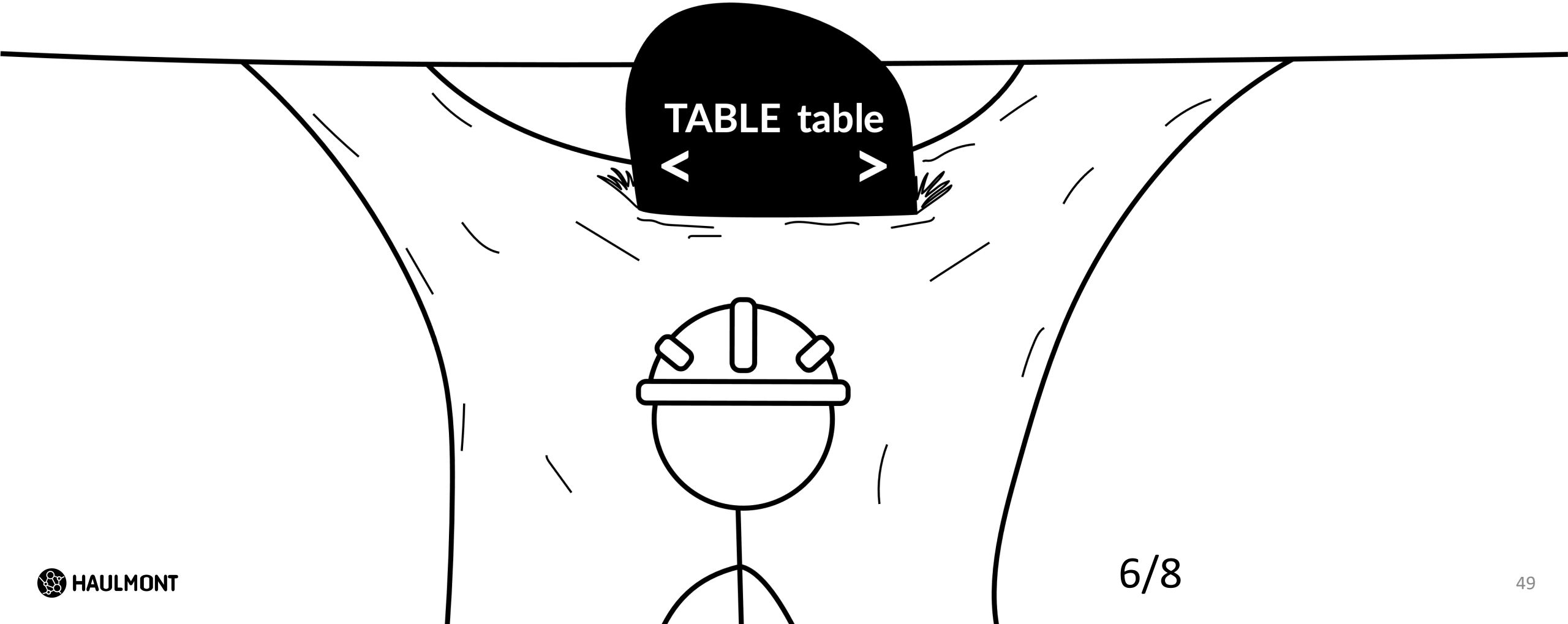
Итого

- Каскадные операции работают со всем графом сущностей
- Любой выход из транзакции -> merge
  - Выборка
- Каскадный merge
  - Много выборок

- Нужно четко понимать, когда нужен Cascade

Имена таблиц

Все не так однозначно



HAULMONT

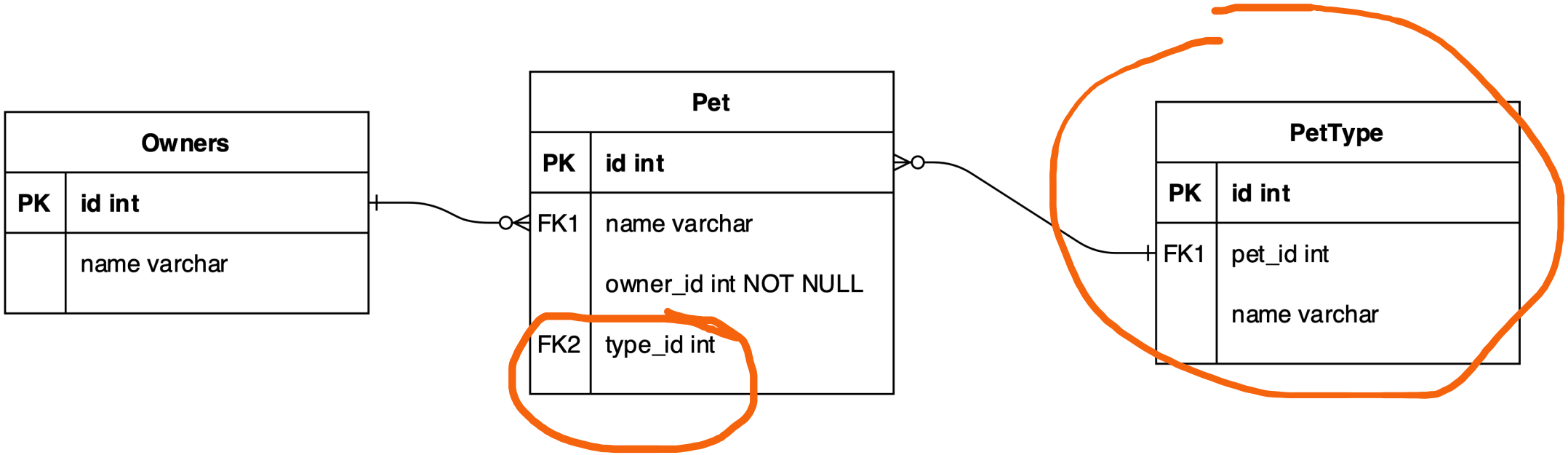
Исходные данные

```java
@Entity
public class PetType {

    @Id
    private Long id; //GET,SET

    private String name; //GET,SET

}
```

```sql
insert into pettype values (1, 'Dog');
insert into pettype values (2, 'Cat');
```

Spring Boot поверх Hibernate

- Минимум аннотаций
- Все равно внутри Spring Boot уже есть Hibernate
- Что будет, если просто убрать лишний код?

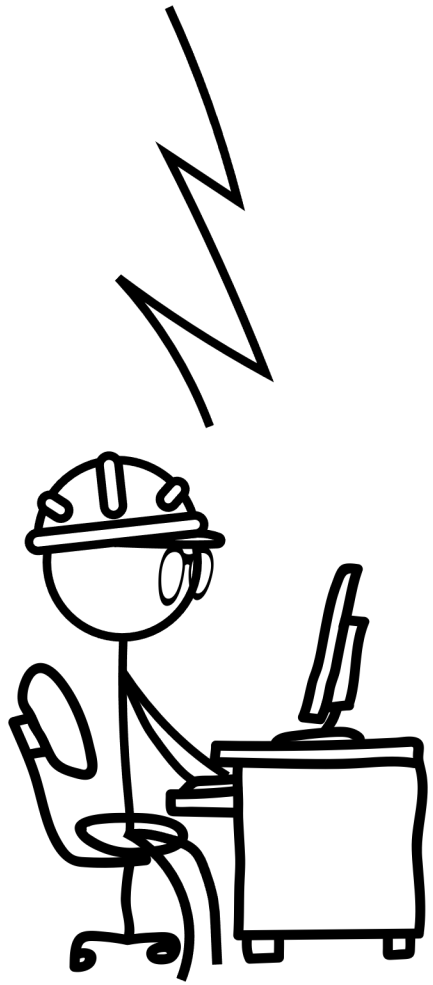


Что произойдет?

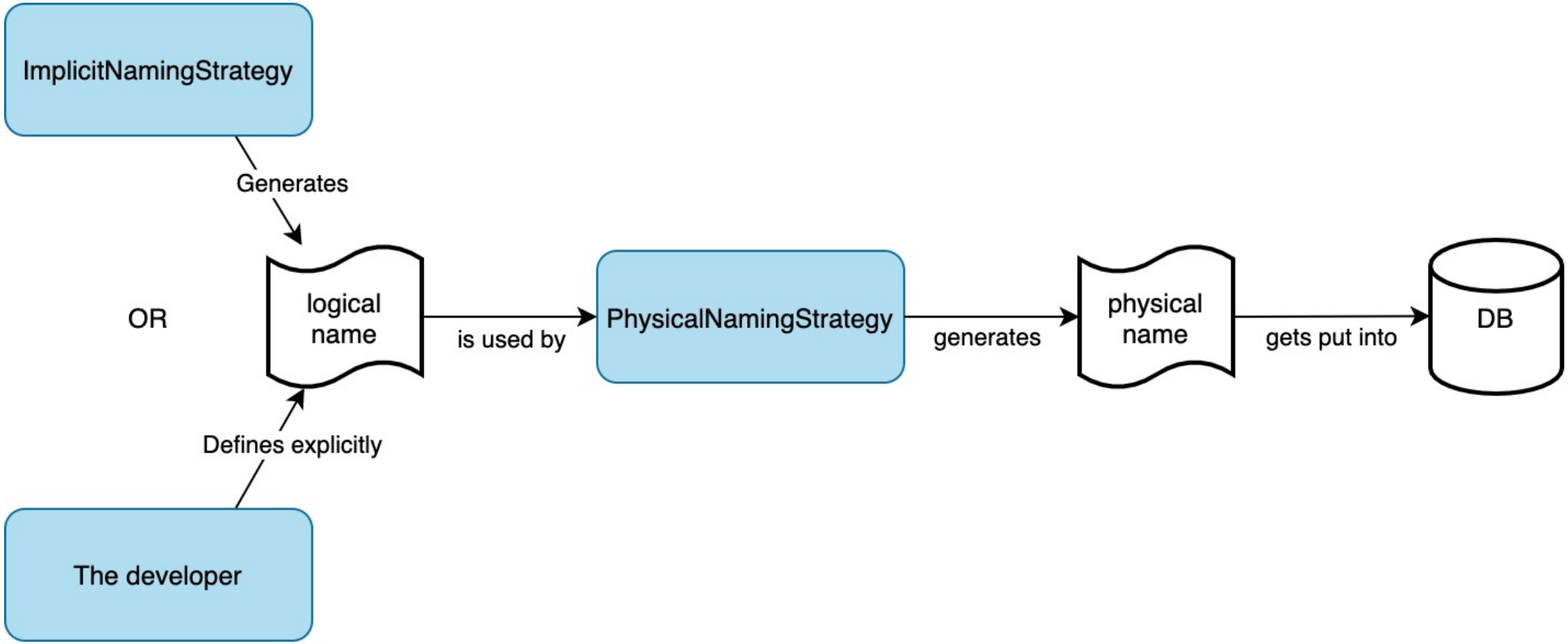
```
petRepository.findById(1L).ifPresent(System.out::println);
```

1. Pet(id=1, name=Fido)
2. LazyInitException
3. HibernateException
4. Ошибка компиляции

Coding Time



Почему так происходит?



Фреймворк против фреймворка

- Hibernate
  - ImplicitNamingStrategy
  - PhysicalNamingStrategy
- Spring Boot
  - SpringImplicitNamingStrategy
  - SpringPhysicalNamingStrategy

```
By default a table is generated corresponding to an entity and bears the
same name as that assigned to the entity (which in turn may have been
defaulted from the name of the entity class)
```

JPA Specs ver 2.1 , ch. 11.2 Object/Relational Metadata Used in Schema Generation, p. 494

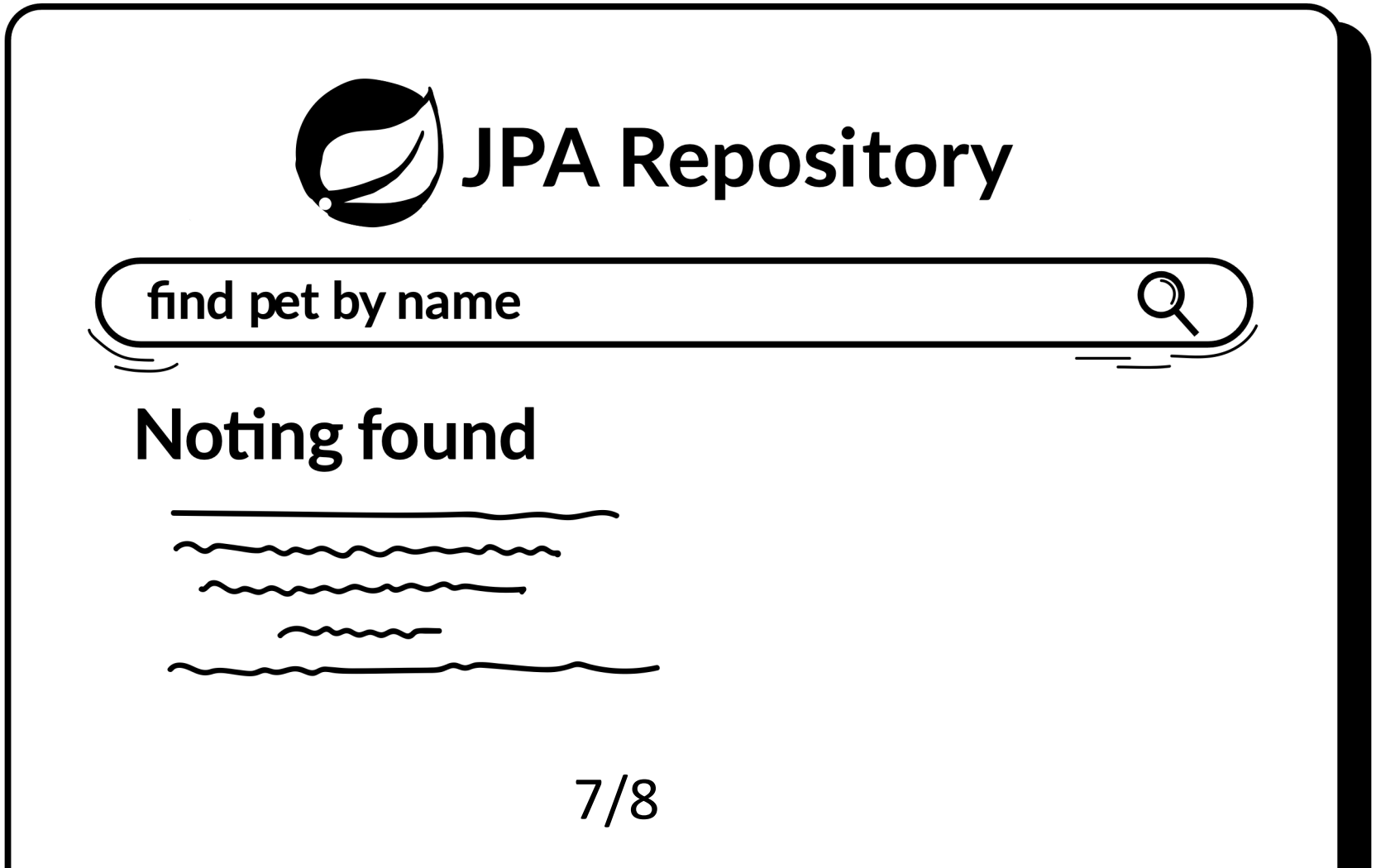
Итого

- Авторы фреймворков могут иметь свое мнение
- Явное всегда лучше неявного
- Для совместимости можно поменять стратегию

```
spring.jpa.hibernate.naming.implicit-strategy
spring.jpa.hibernate.naming.physical-strategy
```

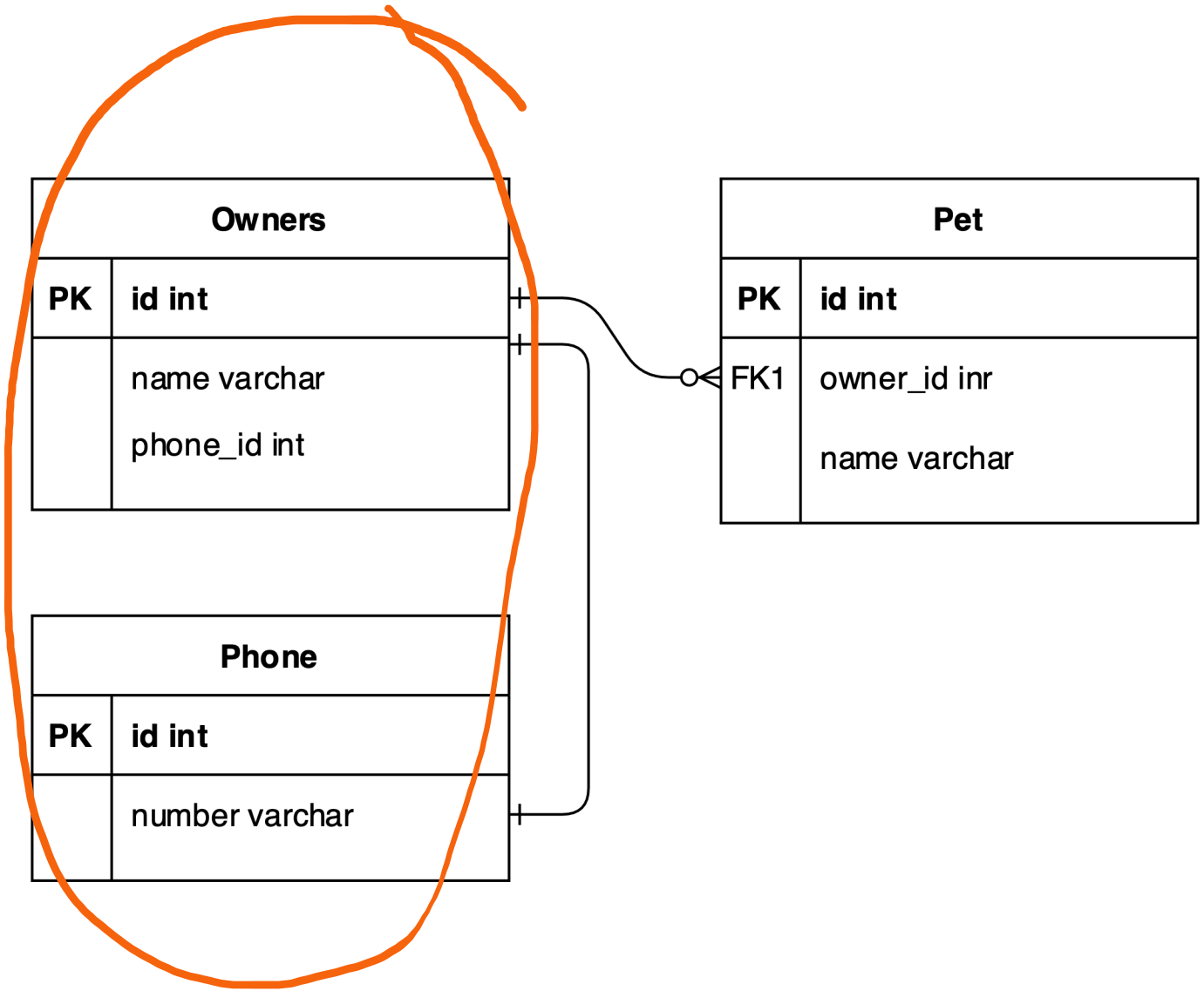
Spring Data JPA

Как писать запросы правильно

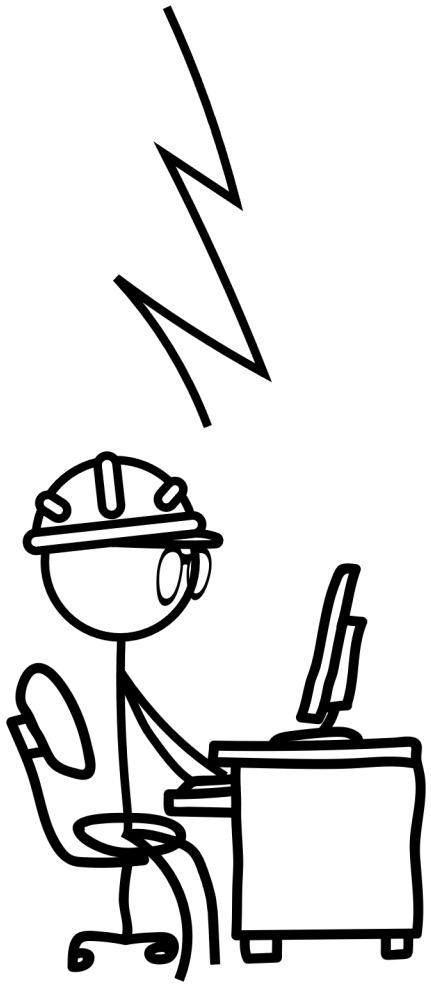


JpaRepository

- Многие пишут запросы
- Boot-time валидация
- Есть ли тут подводные камни?



Coding Time



Почему так происходит?

- 4.4.3. Property Expressions

…

To resolve this ambiguity you can use _ inside your method name to manually define traversal points.

…

Because we treat the underscore character as a reserved character, we strongly advise following standard Java naming conventions (that is, not using underscores in property names but using camel case instead).

Кто за это отвечает?

Spring-Data-Commons: PartTree

```
private static final String KEYWORD_TEMPLATE = "(%s)(?=(\\p{Lu}|\\P{InBASIC_LATIN}))";
private static final String QUERY_PATTERN = "find|read|get|query|search|stream";
private static final String COUNT_PATTERN = "count";
private static final String EXISTS_PATTERN = "exists";
private static final String DELETE_PATTERN = "delete|remove";
private static final Pattern PREFIX_TEMPLATE = Pattern.compile( //
      "^(" + QUERY_PATTERN + "|" + COUNT_PATTERN + "|" + EXISTS_PATTERN + "|" + DELETE_PATTERN + ")((\\p{Lu}.*?))??By");
```

Spring-Data-Commons: Part

```
BETWEEN(2, "IsBetween", "Between"), IS_NOT_NULL(0, "IsNotNull", "NotNull"), IS_NULL(0, "IsNull", "Null"), LESS_THAN(
      "IsLessThan", "LessThan"), LESS_THAN_EQUAL("IsLessThanEqual", "LessThanEqual"), GREATER_THAN("IsGreaterThan",
            "GreaterThan"), GREATER_THAN_EQUAL("IsGreaterThanEqual", "GreaterThanEqual"), BEFORE("IsBefore",
                  "Before"), AFTER("IsAfter", "After"), NOT_LIKE("IsNotLike", "NotLike"), LIKE("IsLike",
                        "Like"), STARTING_WITH("IsStartingWith", "StartingWith", "StartsWith"), ENDING_WITH("IsEndingWith",
                              "EndingWith", "EndsWith"), IS_NOT_EMPTY(0, "IsNotEmpty", "NotEmpty"), IS_EMPTY(0, "IsEmpty",
                                    "Empty"), NOT_CONTAINING("IsNotContaining", "NotContaining", "NotContains"), CONTAINING(
                                          "IsContaining", "Containing", "Contains"), NOT_IN("IsNotIn", "NotIn"), IN("IsIn",
                                                "In"), NEAR("IsNear", "Near"), WITHIN("IsWithin", "Within"), REGEX("MatchesRegex",
                                                      "Matches", "Regex"), EXISTS(0, "Exists"), TRUE(0, "IsTrue", "True"), FALSE(0,
                                                            "IsFalse", "False"), NEGATING_SIMPLE_PROPERTY("IsNot",
                                                                  "Not"), SIMPLE_PROPERTY("Is", "Equals");
```

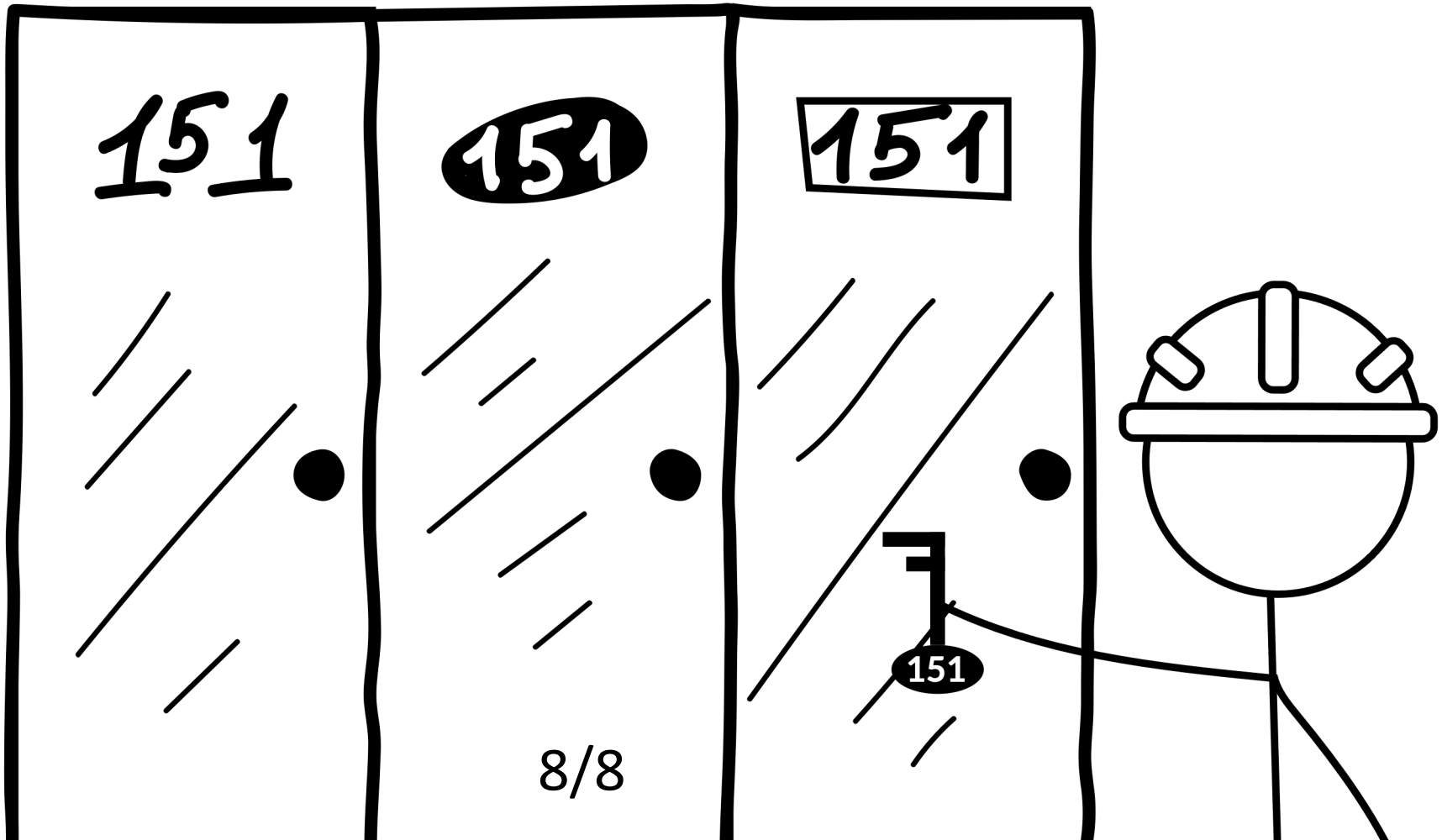
Итого

- В именованиях есть неочевидные тонкости
- Разбор имен делается через RegEx

- Нужно читать документацию
- Или использовать генераторы

Поиск по ID

Можно найти не то, что ищешь



Ищем сущность по ID

- T getById(ID id);
- Optional<T> findById(ID id);
- T getOne(ID id);

А какой поиск выберешь ты?

Ищем сущность по ID

```
ownerRepository.findById(1L).ifPresent(owner -> System.out.println(owner.getId()));

System.out.println(ownerRepository.getById(1L).getId());
```

1. 1
   LazyInitException

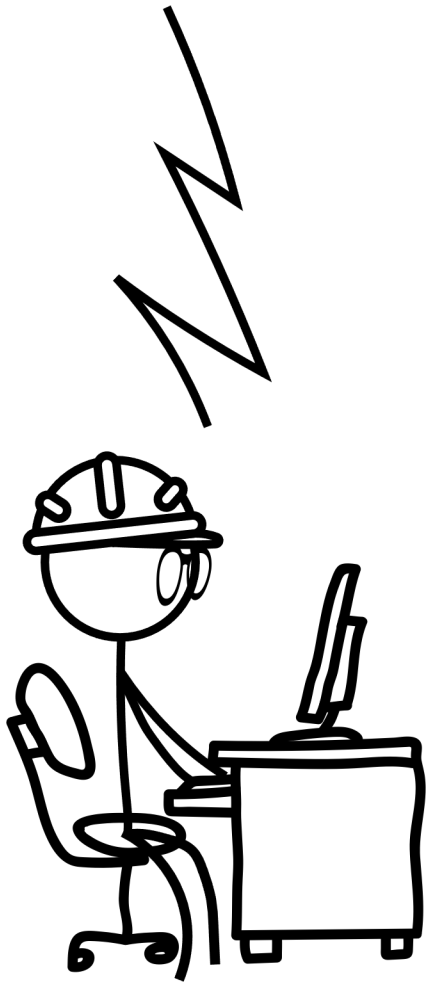
2. 1
   1

3. 1
   IllegalStateException

4. 1
   Owner_Name$Proxy@1DA34

	id	name
1	1	Jane
2	2	John

Coding Time



Ищем сущность по ID

```
ownerRepository.findById(1L).ifPresent(owner -> System.out.println(owner.getName()));

System.out.println(ownerRepository.getById(1L).getName());
```

1. Jane
   LazyInitException

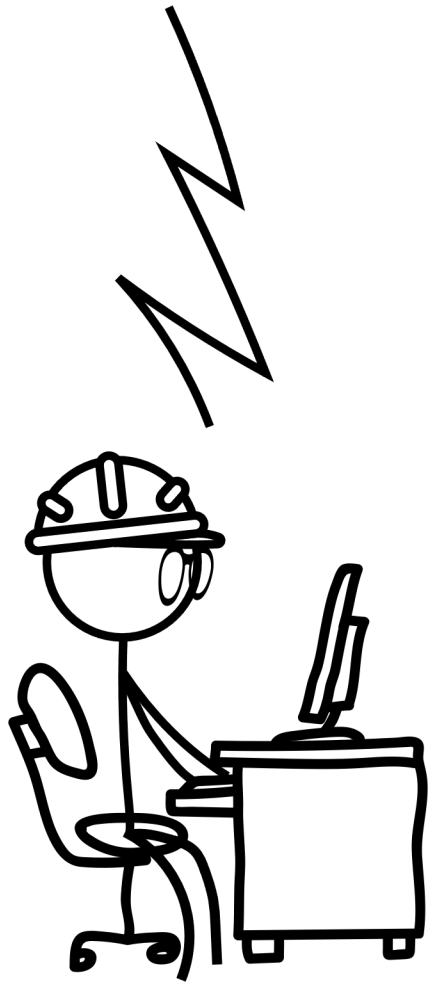
2. Jane
   Jane

3. Jane
   IllegalStateException

4. Jane
   Owner_Name$Proxy@1DA34

	id	name
1	1	Jane
2	2	John

Coding Time



Итого

- Не все названия методов одинаково полезны
- findById – для поиска сущностей
- getById – для простановки ссылок

Совсем итого

- Разработчики фреймворков хотят сделать мир лучше
- Но все учесть невозможно
- Обычно все есть в документации
- Но спецэффекты часто возникают на стыке фреймворков
  - Помогут или инструменты
  - Или учебники
  - Или дебаг

Спасибо за внимание