Unit Of Work

Cenário

```
class ControllerWithoutUnitOfWork {

    @GetMapping( ...value: "/classes/{classId}")
    fun newTeacherToClass(
        @RequestParam teacherName: String,
        @PathVariable classId: Long
    ) {
        val context = DbContextWithoutUnitOfWork()

        context.connect { it: Connection
            val teacherMapper = context.getMapper(Teacher::class.java, it)
            val classMapper = context.getMapper(Clazz::class.java, it)

            val teacher = Teacher(teacherName)
            val clazz = classMapper.find(classId)

            clazz.teacher = teacher

            teacherMapper.insert(teacher)
            classMapper.update(clazz)
        }
    }
}
```

Cenário

```
class ControllerWithoutUnitOfWork {

    @GetMapping( ...value: "/classes/{classId}")
    fun newTeacherToClass(
        @RequestParam teacherName: String,
        @PathVariable classId: Long
    ) {
        val context = DbContextWithoutUnitOfWork()

        context.connect { it: Connection
            val teacherMapper = context.getMapper(Teacher::class.java, it)
            val classMapper = context.getMapper(Clazz::class.java, it)

            val teacher = Teacher(teacherName)
            val clazz = classMapper.find(classId)

            clazz.teacher = teacher

            teacherMapper.insert(teacher)
            classMapper.update(clazz)
        }
    }
}
```

Necessário garantir que todas as alterações efetuadas são refletidas na base de dados

Solução

```
@Controller
class ControllerWithUnitOfWork {

    @GetMapping( ...value: "/classes/{classId}")
    fun newTeacherToClass(
        @RequestParam teacherName: String,
        @PathVariable classId: Long
    ) {
        val context = DbContextWithUnitOfWork()
        UnitOfWork.new()

        val classMapper = context.getMapper(Clazz::class.java)

        val teacher = Teacher(teacherName)
        teacher.markNew()

        val klass = classMapper.find(classId)

        klass.teacher = teacher
        klass.markModified()

        UnitOfWork.current.commit(context)
        UnitOfWork.remove()
    }
}
```

Não há escritas manuais na base de dados

Solução

```
class UnitOfWork {
    private val created = mutableListOf<DomainObject>()
    private val modified = mutableListOf<DomainObject>()
    private val deleted = mutableListOf<DomainObject>()

    fun registerNew(domainObject: DomainObject) {...}

    fun registerDirty(domainObject: DomainObject) {...}

    fun registerDeleted(domainObject: DomainObject) {...}

    fun commit(dbContext: DbContext) {...}

    companion object {
        private val units: ThreadLocal<UnitOfWork> = ThreadLocal()

        fun new() {
            units.set(UnitOfWork())
        }

        fun remove() {
            units.set(null)
        }

        val current: UnitOfWork
            get() = units.get()
    }
```

Como registar?

```
abstract class DomainObject {
    fun markNew() {
        UnitOfWork.current.registerNew( domainObject: this)
    }

    fun markModified() {
        UnitOfWork.current.registerDirty( domainObject: this)
    }

    fun markDeleted() {
        UnitOfWork.current.registerDeleted( domainObject: this)
    }
}
```

Como registar?

```
@Controller
class ControllerWithUnitOfWork {

    @GetMapping( ...value: "/classes/{classId}")
    fun newTeacherToClass(
        @RequestParam teacherName: String,
        @PathVariable classId: Long
    ) {
        DbContextWithUnitOfWork().use { it: DbContextWithUnitOfWork
            val classMapper = it.getMapper(Clazz::class.java)

            val teacher = Teacher(teacherName)
            teacher.markNew()

            val klass = classMapper.find(classId)

            klass.teacher = teacher
            klass.markModified()

            it.commit()
        }
    }
}
```

Registo manual

Como registar?

```
abstract class DomainObject {
    init {
        markNew()
    }

    fun markNew() {
        UnitOfWork.current.registerNew( domainObject: this)
    }

    fun markModified() {
        UnitOfWork.current.registerDirty( domainObject: this)
    }

    fun markDeleted() {
        UnitOfWork.current.registerDeleted( domainObject: this)
    }
}
```

Registo automático

Como registar?

```
class Clazz(
    private val students: MutableList<Student> = mutableListOf(),
    teacher: Teacher
) : DomainObject() {

    var teacher: Teacher = teacher
        set(value) {
            markModified()
            field = value
        }

    fun addStudent(student: Student) {
        markModified()
        students.add(student)
    }

    fun removeStudent(student: Student) {
        markModified()
        students.remove(student)
        student.markDeleted()
    }
}
```

Registo automático

API mais conveniente

```
class DbContextWithUnitOfWork : AbstractDbContext(), Closeable {

    init {
        UnitOfWork.new()
    }

    fun commit() {
        UnitOfWork.current.commit( dbContext: this)
    }

    override fun close() {
        UnitOfWork.remove()
    }
}
```

```
@Controller
class ControllerWithUnitOfWork {

    @GetMapping( ...value: "/classes/{classId}")
    fun newTeacherToClass(
        @RequestParam teacherName: String,
        @PathVariable classId: Long
    ) {
        DbContextWithUnitOfWork().use { it: DbContextWithUnitOfWork
            val classMapper = it.getMapper(Clazz::class.java)

            val teacher = Teacher(teacherName)

            val klass = classMapper.find(classId)

            klass.teacher = teacher

            it.commit()
        }
    }
}
```

Commit

Qual é a ordem de escrita?

Commit

Qual é a ordem de escrita?

```
private val INSERTION_ORDER = listOf(
    Course::class.java,
    Teacher::class.java,
    Student::class.java,
    Clazz::class.java
)
```

Geralmente, a ordem de remoção será a inversa da ordem de inserção

Recomendações de leitura

Martin Fowler - Patterns of Enterprise Application Architecture

Código

Github - github.com/JorgePBrown/UnitOfWorkDemo