Spring PetClinic Coding Rules

This document describes coding rules for the Spring PetClinic.

The rules defined in this and all included documents are for demonstration purposes only and work in progress. For rendering of embedded diagrams Graphviz[1] needs to be installed and available via the system path.

1. CI Build

The following rule groups are executed during a build:

- Section 2, “Structural Rules And Reports”
- Section 3, “JPA Model”
- Section 4, “Management”
- Section 5, “Test Rules”

2. Structural Rules And Reports

The following constraints are verified:

- A controller can only have dependencies to services (e.g. direct usage of repositories is not allowed).
- A service can only have dependencies to repositories or other services (e.g. usage of controllers is not allowed).
- A repository can only have dependencies to other repositories.
- There must be no direct dependencies between Spring component implementations, i.e. only dependencies to interfaces are allowed.
- Package cycles

The following GraphML reports are created:

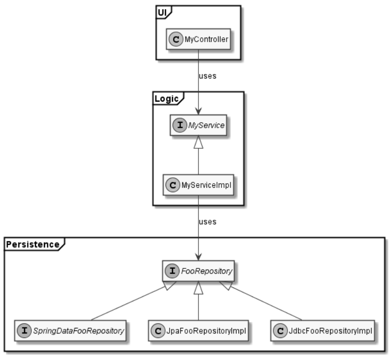
- Actual usage dependencies between the application layers, i.e. controller, service and repository implementations.

[1] http://www.graphviz.org

- Dependencies between packages and their contained types.

2.1. Layers And Elements

The application consists of the following layers:



- UI
  - Controller implementations
- Logic
  - Service interfaces and implementations
- Persistence
  - Repository interfaces and implementations
    - Spring Data (interface declaration only)
    - JPA
    - JDBC

2.2. Constraints

The following sections describe restrictions on dependencies between these layers.

General

There must be no direct dependencies between Spring component implementations, i.e. only dependencies to interfaces are allowed.

```
MATCH
  (type:Spring:Component)-[:DECLARES]->(:Method)-[:INVOKES]->(:Method)<-
[:DECLARES]-(otherType:Spring:Component)
WHERE
  type <> otherType
  and not otherType:Interface
RETURN DISTINCT
  type as Dependent, otherType as InvalidDependency
```

UI

The server side UI implementation is based on Spring MVC controllers:

Example controller.

```
@Controller
@RequestMapping("/resource")
public class MyController {

    private final MyService service;

    @Autowired
    public MyController(MyService myService) {
        this.myService = myService;
    }

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String get() {
    }
}
```

A controller can only have dependencies to services (e.g. direct usage of repositories is not allowed).

```
MATCH
  (type:Spring:Controller)-[:USES]->(otherType:Spring:Component)
WHERE NOT
  otherType:Service
RETURN DISTINCT
  type as InvalidController, otherType as Dependency
```

Logic

The business logic consists of services which are defined by interfaces and implemented by classes annotated with @Service:

Example service interface.

```
public interface MyService {
    Collection<Foo> findFoos() throws DataAccessException;
}
```

Example service implementation.

```
@Service
public class MyServiceImpl implements MyService {

    private FooRepository fooRepository;

    @Autowired
    public ServiceImpl(FooRepository fooRepository) {
        this.fooRepository = fooRepository;
    }
}
```

A service can only have dependencies to repositories or other services (e.g. usage of controllers is not allowed).

```
MATCH
  (type:Spring:Service)-[:USES]->(otherType:Spring:Component)
WHERE NOT (
  otherType:Service
  or otherType:Repository
)
RETURN DISTINCT
  type as InvalidService, otherType as Dependency
```

Persistence

Repositories provide access to the database and are defined by interfaces (one per model element):

Example repository interface.

```
public interface FooRepository {
    Collection<Foo> findFoos() throws DataAccessException;
}
```

There are three options to provide or implement a repository:

Example Spring Data repository.

```
public interface SpringDataFooRepository extends FooRepository,
 Repository<Foo, Integer> {

    @Override
    @Query("SELECT foo FROM Foo ORDER BY foo.name")
    List<Foo> findFoos() throws DataAccessException;
}
```

Example JPA repository.

```
@Repository
public class JpaFooRepositoryImpl implements FooRepository {

    @PersistenceContext
    private EntityManager em;

    @Override
    @SuppressWarnings("unchecked")
    public List<Foo> findFoos() {
        return this.em.createQuery("SELECT foo FROM Foo ORDER BY
 foo.name").getResultList();
    }
```

Example JDBC repository.

```
@Repository
public class JdbcFooRepositoryImpl implements Repository {

    private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    @Autowired
    public JdbcRepositoryImpl(DataSource dataSource) {
        this.namedParameterJdbcTemplate = new
 NamedParameterJdbcTemplate(dataSource);
    }

    @Override
    public List<Foo> findFoos() throws DataAccessException {
        Map<String, Object> params = new HashMap<>();
        return this.namedParameterJdbcTemplate.query(
            "SELECT id, name FROM foo ORDER BY name",
            params,
            BeanPropertyRowMapper.newInstance(Foo.class));
    }
```

A repository can only have dependencies to other repositories.

```
MATCH
  (type:Spring:Repository)-[:USES]->(otherType:Spring:Component)
WHERE NOT
  otherType:Repository
RETURN DISTINCT
  type as InvalidRepository, otherType as Dependency
```

2.3. Concepts

Creates a USES relation between application layer items, i.e. spring components.

```
MATCH
  (type:Spring:Component)-[:DECLARES]->(:Method)-[:INVOKES]-
>(:Method)<-[:DECLARES]-(:Type:Interface)<-[:IMPLEMENTS|EXTENDS*]-
(otherType:Spring:Component)
WHERE
  type <> otherType
MERGE
  (type)-[:USES]->(otherType)
RETURN
  type as Dependent, collect(distinct otherType.fqn) as Dependencies
```

2.4. Reports

GraphML reports may be viewed using yEd[2]. After opening a file you should apply a layout, e.g. menu:Layout[Hierarchical].

Actual usage dependencies between the application layers, i.e. controller, service and repository implementations.

[2] http://www.yworks.com/en/products/yfiles/yed/

```
MATCH
  (type:Spring:Component)-[uses:USES]->(otherType:Spring:Component)
RETURN
  type as Dependent,
  uses as Uses,
  otherType as Dependency
```

Dependencies between packages and their contained types.

```
MATCH
  (package:Package)-[:CONTAINS]->(type:Type)
OPTIONAL MATCH
  (type)-[dependsOn:DEPENDS_ON]->(:Type)
RETURN {
  role : "graph",
  parent : package,
  nodes : collect(type),
  relationships : collect(dependsOn)
} as TypesPerPackage
```

```
//zeige alle Klassenbeziehungen
OPTIONAL MATCH
  (type)-[dependsOn:DEPENDS_ON]->(t2:Type)
WHERE
(type.fqn CONTAINS "petclinic" AND t2.fqn CONTAINS "petclinic")
AND (NOT type.name ENDS WITH "Test" AND NOT type.name ENDS WITH "Test")
RETURN {
  role : "graph",
  parent : type,
  nodes : collect(type),
  relationships : collect(dependsOn)
} as Types
```

3. JPA Model

The following constraints are verified:

- All JPA entities must be located in packages named "model".

3.1. Constraints

All JPA entities must be located in packages named "model".

```
MATCH
  (package:Package)-[:CONTAINS]->(entity:Jpa:Entity)
WHERE
  package.name <> "model"
RETURN
  entity AS EntityInWrongPackage
```

4. Management

The following reports are created:

- All managed resources including their attributes and operations.

4.1. Reports

All managed resources including their attributes and operations.

```
MATCH
  (managedResource:Type)-[:DECLARES]->(member)
WHERE
  member:ManagedAttribute
  or member:ManagedOperation
RETURN
  managedResource as ManagedResource, collect(member.name) as Member
```

5. Test Rules

The following test rules apply:

- All test methods must perform at least one assertion (within a call hierarchy of max. 3 steps).
- All @Ignore annotations must provide a message

5.1. Constraints

All test methods must perform at least one assertion (within a call hierarchy of max. 3 steps).

```
MATCH
  (testType:Test:Type)-[:DECLARES]->(testMethod:Test:Method)
WHERE
```

```
  NOT (testMethod)-[:INVOKES*..3]->(:Method:Assert)
RETURN
  testType AS DeclaringType,
  testMethod AS Method
```

5.2. Concepts

The following framworks are used to verify assertions:

- AssertJ
- Spring Web Test

Mark all assertThat methods of 'org.assertj.core.api.Assertions' with "AssertJ" and "Assert".

```
MATCH
  (assertType:Type)-[:DECLARES]->(assertMethod)
WHERE
  assertType.fqn = 'org.assertj.core.api.Assertions'
  and assertMethod.signature =~ '.* assertThat.*'
SET
  assertMethod:AssertJ:Assert
RETURN
  assertMethod
```

Mark method "org.springframework.test.web.servlet.ResultActions#andExpect" as "Spring" and "Assert".

```
MATCH
  (assertType:Type)-[:DECLARES]->(assertMethod)
WHERE
  assertType.fqn = 'org.springframework.test.web.servlet.ResultActions'
  and assertMethod.signature =~ '.* andExpect.*'
SET
  assertMethod:Spring:Assert
RETURN
  assertMethod
```

6. Spring MVC Rules

This document defines concepts related to the Spring MVC Framework.

6.1. Concepts

Labels all types annotated with "org.springframework.stereotype.Service" with "Spring", "Component" and "Service".

```
MATCH
  (service:Type)-[:ANNOTATED_BY]->()-[:OF_TYPE]->(annotationType:Type)
WHERE
  annotationType.fqn = "org.springframework.stereotype.Service"
SET
  service:Spring:Component:Service
RETURN
  service as Service
```

Labels all types annotated with "org.springframework.stereotype.Controller" with "Spring", "Component" and "Controller".

```
MATCH
  (controller:Type)-[:ANNOTATED_BY]->()-[:OF_TYPE]-
>(annotationType:Type)
WHERE
  annotationType.fqn = "org.springframework.stereotype.Controller"
SET
  controller:Spring:Component:Controller
RETURN
  controller as Controller
```

7. Spring Data Rules

This document defines concepts related to the Spring Data Framework.

7.1. Concepts

Returns all repositories.

```
MATCH
  (repository:Spring:Repository)
RETURN
  repository as Repository
```

Labels all types annotated with "org.springframework.stereotype.Repository" with "Spring", "Component" and "Repository".

```
MATCH
  (repository:Type)-[:ANNOTATED_BY]->()-[:OF_TYPE]-
>(annotationType:Type)
WHERE
  annotationType.fqn = "org.springframework.stereotype.Repository"
SET
  repository:Spring:Component:Repository
RETURN
  repository as Repository
```

Labels all types implementing "org.springframework.data.repository.Repository" with "Spring", "Component" and "Repository".

```
MATCH
  (repository:Type)-[:EXTENDS|IMPLEMENTS*]->(superType:Type)
WHERE
  superType.fqn in [
    "org.springframework.data.repository.Repository"
  ]
SET
  repository:Spring:Component:Repository
RETURN
  repository as Repository
```

8. Spring JMX Rules

This document defines concepts related to the Spring JMX Framework.

8.1. Concepts

Labels all types annotated with "org.springframework.jmx.export.annotation.ManagedResource" with "Spring" and "ManagedResource".

```
MATCH
  (managedResource:Type)-[:ANNOTATED_BY]->()-[:OF_TYPE]-
>(annotationType:Type)
WHERE
```

```
  annotationType.fqn =
 "org.springframework.jmx.export.annotation.ManagedResource"
SET
  managedResource:Spring:ManagedResource
RETURN
  managedResource as ManagedResource
```

Labels all methods annotated with "org.springframework.jmx.export.annotation.ManagedAttribute" with "Spring" and "ManagedAttribute".

```
MATCH
  (managedAttribute:Method)-[:ANNOTATED_BY]->()-[:OF_TYPE]-
>(annotationType:Type)
WHERE
  annotationType.fqn =
 "org.springframework.jmx.export.annotation.ManagedAttribute"
SET
  managedAttribute:Spring:ManagedAttribute
RETURN
  managedAttribute as ManagedAttribute
```

Labels all methods annotated with "org.springframework.jmx.export.annotation.ManagedAttribute" with "Spring" and "ManagedOperation".

```
MATCH
  (managedOperation:Method)-[:ANNOTATED_BY]->()-[:OF_TYPE]-
>(annotationType:Type)
WHERE
  annotationType.fqn =
 "org.springframework.jmx.export.annotation.ManagedOperation"
SET
  managedOperation:Spring:ManagedOperation
RETURN
  managedOperation as ManagedOperation
```