Loeng 6

Spring Core raamistik

Kordamine

- Mis on javax.sql.DataSource?

Spring raamistikud

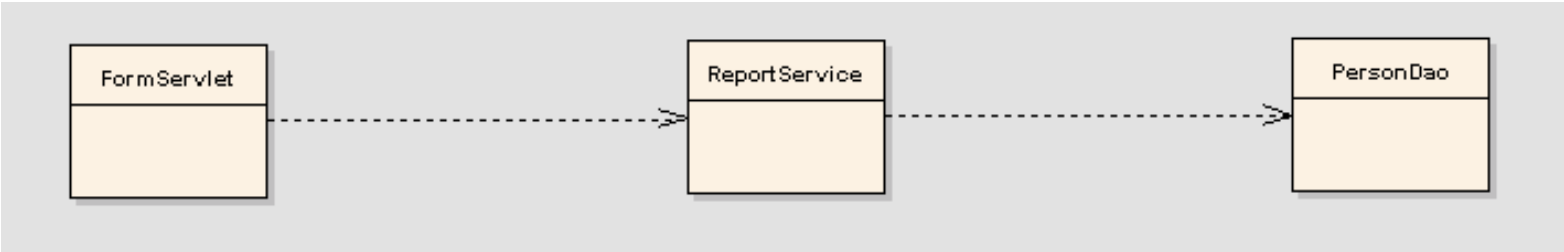
- Spring Core
- Spring MVC
- Spring Security
- Spring Data
- Spring Boot
- Spring Cloud
- Spring Integration
- ...

Spring Core

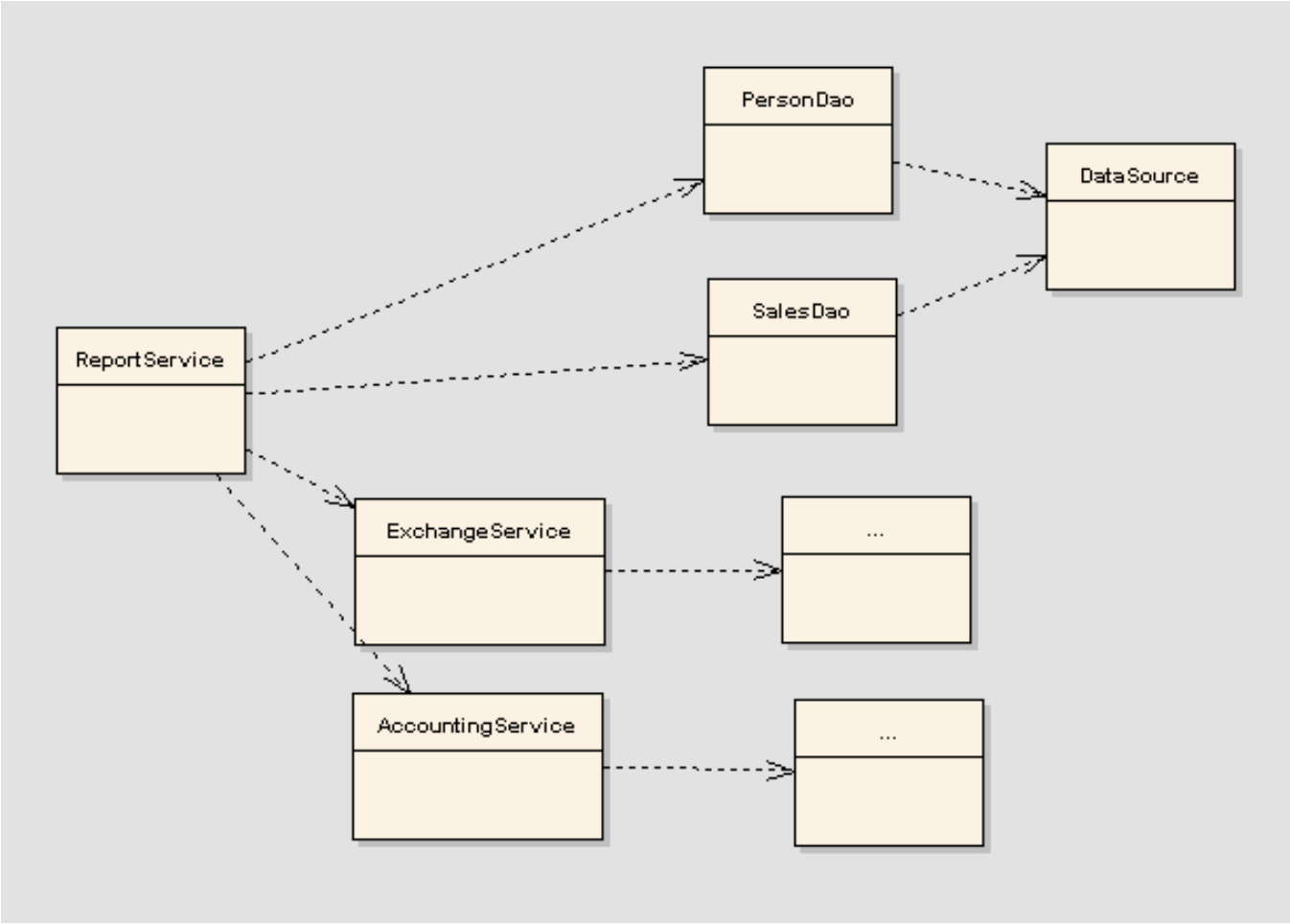
```
implementation group: 'org.springframework',
        name: 'spring-context',
        version: '5.2.9.RELEASE'

implementation group: 'org.springframework',
        name: 'spring-jdbc',
        version: '5.2.9.RELEASE'

implementation group: 'org.springframework',
        name: 'spring-aop',
        version: '5.2.9.RELEASE'
```

Kihiline arhitektuur

Veebikiht | Teenuskiht | Andmekiht

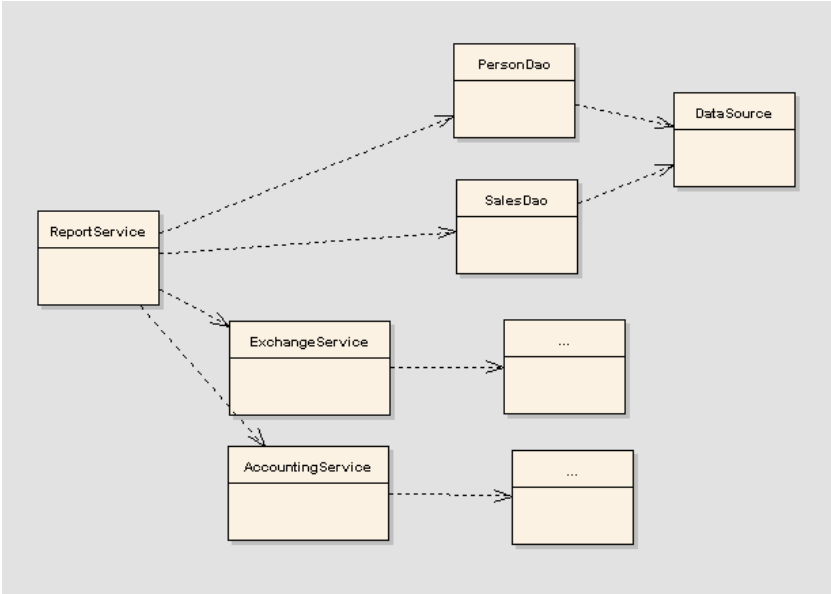


Spring Core raamistik



Mida Spring annab?

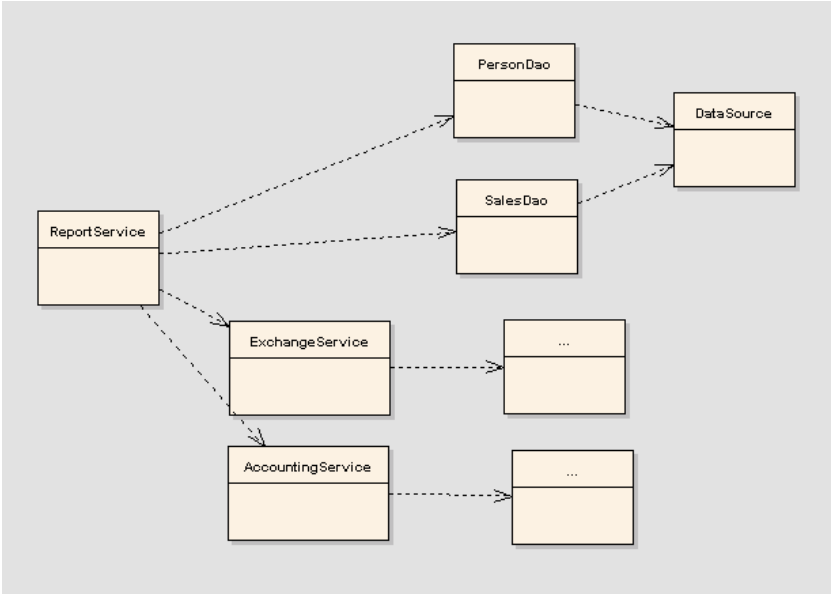
- Paindlikkus
- Mugavus
- Aspect Oriented Programming (AOP)
- Tava



Paindlikkus

New

```
public class PersonDao {

    private DataSource ds = new SomeDataSource();

    // ...
}
```

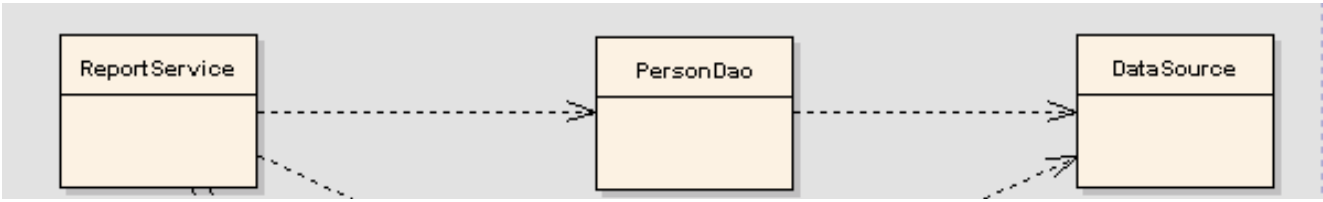


Setters

```
public class PersonDao {

    private DataSource ds;

    public void setDataSource(DataSource dataSource) {
        this.ds = dataSource;
    }

}
```

Loomise meetodid

```
ReportService service = getReportService();

private ReportService getReportService() {
    PersonDao personDao = new PersonDao();
    personDao.setDataSource(getDataSource());
    ReportService reportService = new ReportService();
    reportService.setPersonDao(personDao);
    return reportService;
}

private DataSource getDataSource() {
    DataSource dataSource = ...;
    dataSource.setDriverClassName("org.postgresql.Driver");
    dataSource.setUsername("...");
    ...
}
```

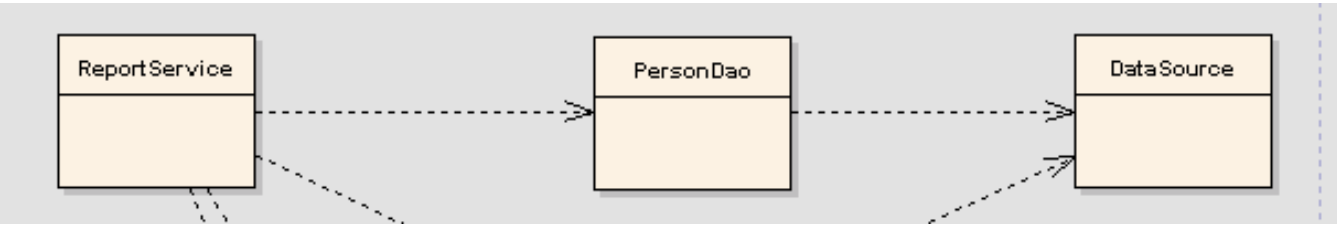


Spring Core

```
ConfigurableApplicationContext ctx =
        new AnnotationConfigApplicationContext(Config.class);


PersonDao dao = ctx.getBean(PersonDao.class);
```

Spring

```
@Bean
public PersonDao getPersonDao(DataSource dataSource) {
    return new PersonDao(dataSource);
}

@Bean
public DataSource dataSource() {
   DriverManagerDataSource ds = new DriverManagerDataSource();

   ds.setDriverClassName("org.postgresql.Driver");
   ds.setUsername("...");
   ds.setPassword("...");
   ds.setUrl("...");

   return ds;
}
```



Demo (bean-i küsimine)

@ComponentScan

```
@Configuration
@ComponentScan(basePackages = {"mypackage"})
public class Config {

}
```

```
package mypackage;

@Component
public class ReportService {
    ...
}
```

```
package mypackage;

@Component
public class PersonDao {
    ...
}
```

Demo (@ComponentScan)

Sõltuvused

```
@Component
public class ReportService {

    private PersonDao personDao;

    public ReportService(PersonDao personDao) {
        this.personDao = personDao;
    }
}
```

Option 1. Defined in configuration file

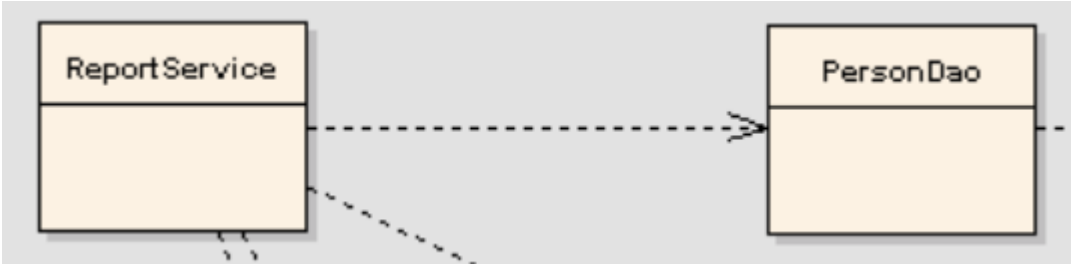
```
@Bean
public PersonDao getPersonDao() {
    return new PersonDao();
}
```

Option 2. Found with component scan

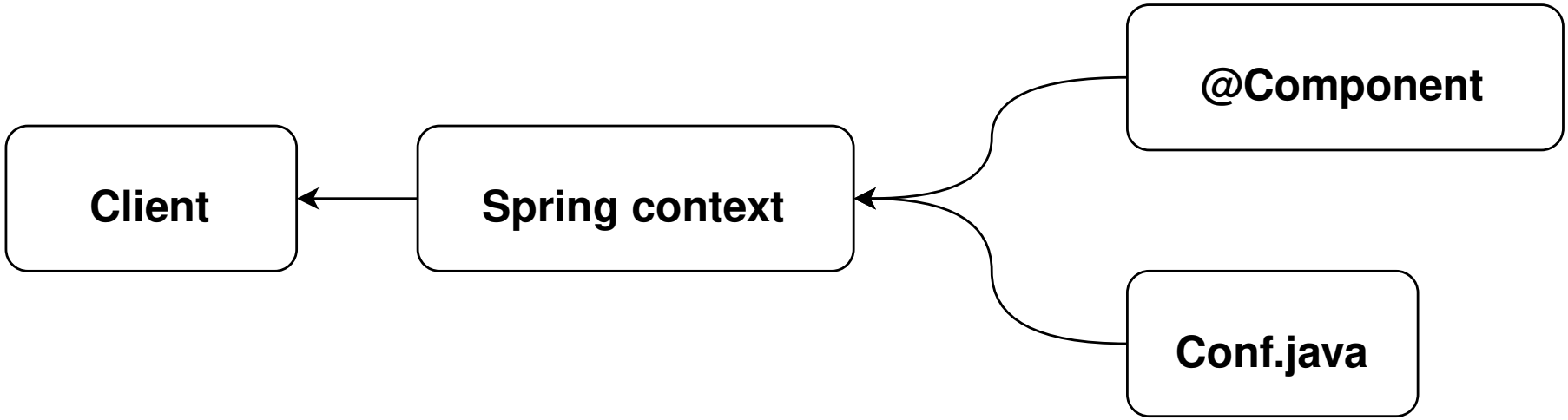
```
@Component
public class PersonDao {

}
```

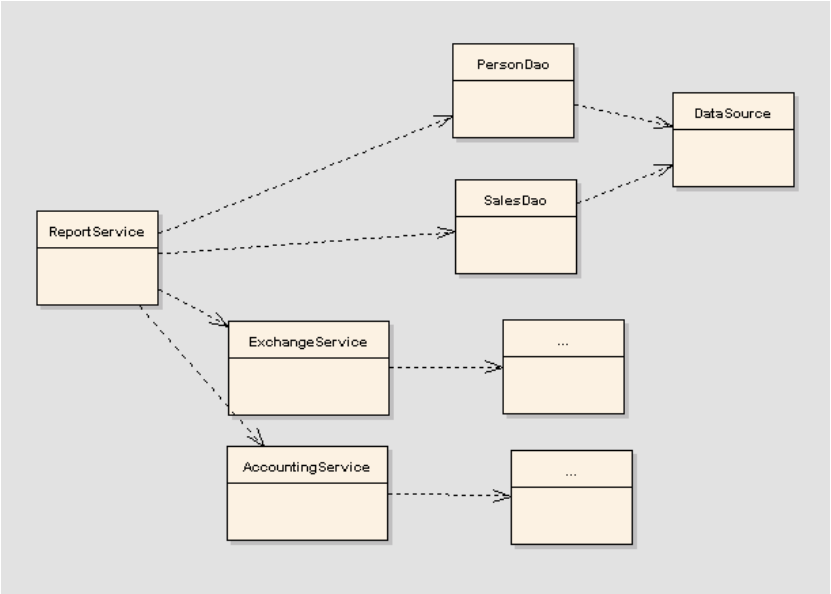
Termin: Dependency Injection (DI)

```
@Component
public class ReportService {

    private PersonDao personDao;

    public ReportService(PersonDao personDao) {
        this.personDao = personDao;
    }
}
```



Spring


@Component
Client
Spring context
Conf.java


PersonDao
DataSource
SalesDao
ReportService
ExchangeService
...
AccountingService
...

Annotatsioonid

- @Service - teenusele
- @Repository – Dao-le
- @Component – ülejäänutele

```
@Component
public class ReportService {
```

```
@Component
public class PersonDao {
```

```
@Service
public class ReportService {
```

```
@Repository
public class PersonDao {
```

Paindlikkus

- Leitakse sobiv implementatsioon

```
class RealPersonDao implements PersonDao { ... };
```

või

```
class TestPersonDao implements PersonDao { ... };
```

```
PersonDao dao = ctx.getBean(PersonDao.class);
```

Profiilid

```
@Configuration
@Profile("production")
public class ProductionConfig {
```

```
@Configuration
@Profile("test")
public class TestConfig {
```

```
ConfigurableApplicationContext ctx =
        new AnnotationConfigApplicationContext(
                TestConfig.class, ProductionConfig.class);

PersonDao dao = ctx.getBean(PersonDao.class);
```

Profiili valitakse keskonna muutuja järgi

```
Win> set SPRING_PROFILES_ACTIVE=test

Mac> export SPRING_PROFILES_ACTIVE=test
```

Demo (implementatsiooni valimine)

Profiilide kasutus

- Erinev andmebaas
- Ühenduste puulimine
- Teavituste saatmine
- Logimine
- Ligipääs
- ...

JdbcTemplate

```
@Configuration
public class SpringConfig {

    @Bean
    public JdbcTemplate getTemplate(DataSource ds) {
        return new JdbcTemplate(ds);
    }

}
```

JdbcTemplate (sisestus)

```
@Repository
public class PersonDao {

    public JdbcTemplate template;

    public PersonDao(JdbcTemplate template) {
        this.template = template;
    }

    public void save(Person person) {
        String sql = "INSERT INTO person (name, age) values (?, ?)";

        template.update(sql, person.getName(), person.getAge());
    }
}
```

JdbcTemplate (tagastusega)

```
@Repository
public class PersonDao {

    ...

    public String getPersonName(Integer id) {
        return template.queryForObject(
                "select name from person where id = ?",
                new Object[] { id }, String.class);
    }
```

JdbcTemplate (objekti tagastusega)

```
public List<Person> findAllPersons() {
    return template.query("select id, name from person",
            new PersonMapper());
}

private class PersonMapper implements RowMapper<Person> {
    public Person mapRow(ResultSet rs, int rowNum)
            throws SQLException {

        Person person = new Person();
        person.setId(rs.getLong("id"));
        person.setName(rs.getString("name"));
        ...

        return person;
    }
}
```

JdbcTemplate (objekti tagastusega)

```
public List<Person> findAllPersonsAutoMapper() {
    return template.query("select id, name from person",
            new BeanPropertyRowMapper<Person>(Person.class));
}
```

JdbcTemplate (ridade töötlus)

```
private class PersonRowHandler implements RowCallbackHandler {
    List<Person> result = ...

    public void processRow(ResultSet rs) throws SQLException {

        String name = rs.getString("name");

    }

    public List<Person> getResult() { ...
}
```

JdbcTemplate (ridade töötlus)

```
var handler = new PersonRowHandler();

template.query(sql, handler);

handler.getResult();
```

SimpleJdbcInsert

```
var data = Map.of("name", "Alice",
                  "age", 20);

Number id = new SimpleJdbcInsert(template)
        .withTableName("person")
        .usingGeneratedKeyColumns("id")
        .executeAndReturnKey(data);
```

SimpleJdbcInsert

```
var data = new BeanPropertySqlParameterSource(person);

Number id = new SimpleJdbcInsert(template)
        .withTableName("person")
        .usingGeneratedKeyColumns("id")
        .executeAndReturnKey(data);
```

Skeemi loomine

```
@Bean
public JdbcTemplate getTemplate(DataSource dataSource) {

    var populator = new ResourceDatabasePopulator(
            new ClassPathResource("schema.sql"),
            new ClassPathResource("data.sql"));

    DatabasePopulatorUtils.execute(populator, dataSource);

    return new JdbcTemplate(dataSource);
}
```

Tüüp vs. nimi

```
ReportService reportService =
        (ReportService) context.getBean("reportService");
```

```
ReportService reportService =
        context.getBean(ReportService.class);
```

Sama tüüpi klassid

```
@Configuration
public class Config {

    @Bean(name = "postgreDs")
    public DataSource postgreDataSource() { ...

    @Bean(name = "mongoDs")
    public DataSource mongoDataSource() { ...
```

Sama tüüpi klassid

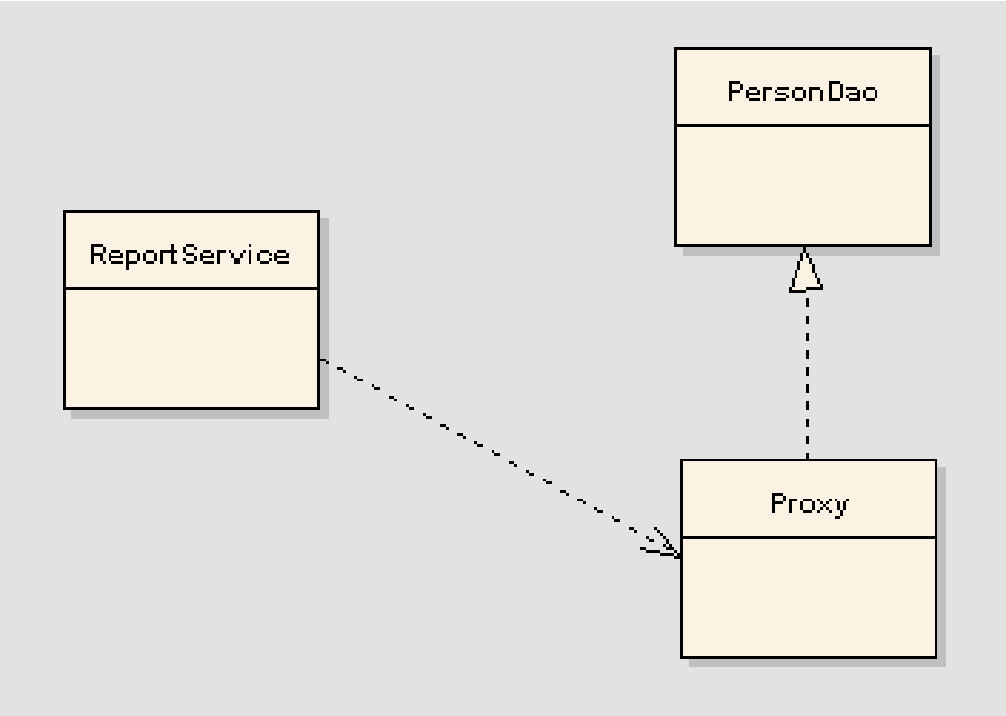
```
@Repository
public class Dao {

    private DataSource postgreDs;

    private DataSource mongoDs;

    public Dao(@Qualifier("postgreDs") DataSource postgreDs,
               @Qualifier("mongoDs") DataSource mongoDs) {

        this.postgreDs = postgreDs;
        this.mongoDs = mongoDs;
    }

}
```

Aspect Oriented Programming (AOP)

Logimine

```
public class PersonDao {
    private static final String log = ...;

    public Person getPerson(Long personId) {
        log.debug("PersonDao.getPerson(): person id: " + personId);

        // ...

    }

    // palju teisi meetodeid
}
```

AOP



execution(public * example.PersonDao.*(..))

Demo (AOP)

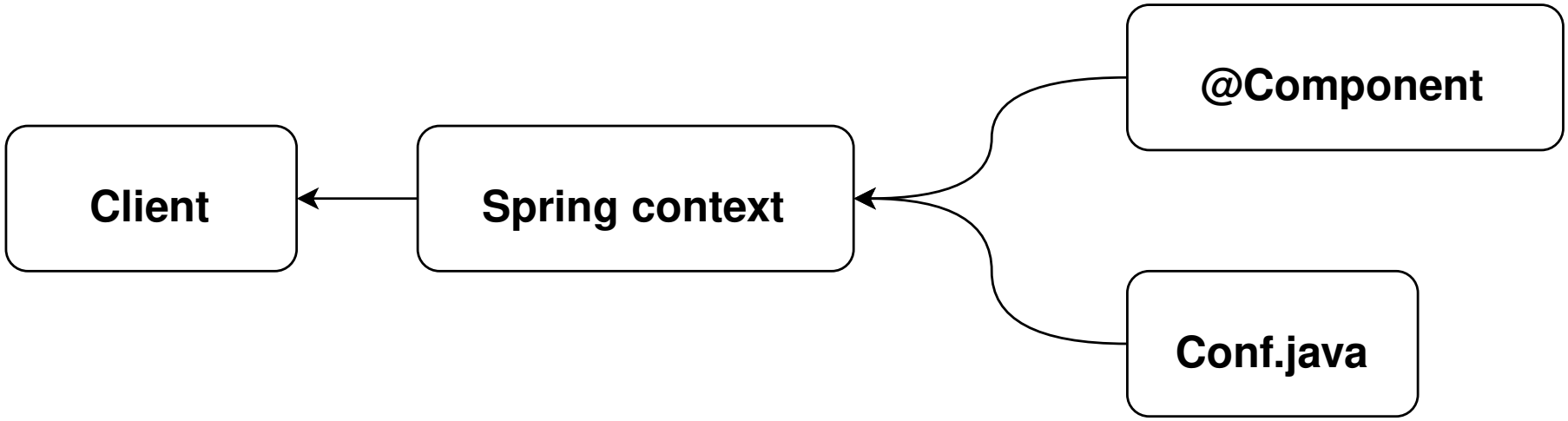
Aop näide

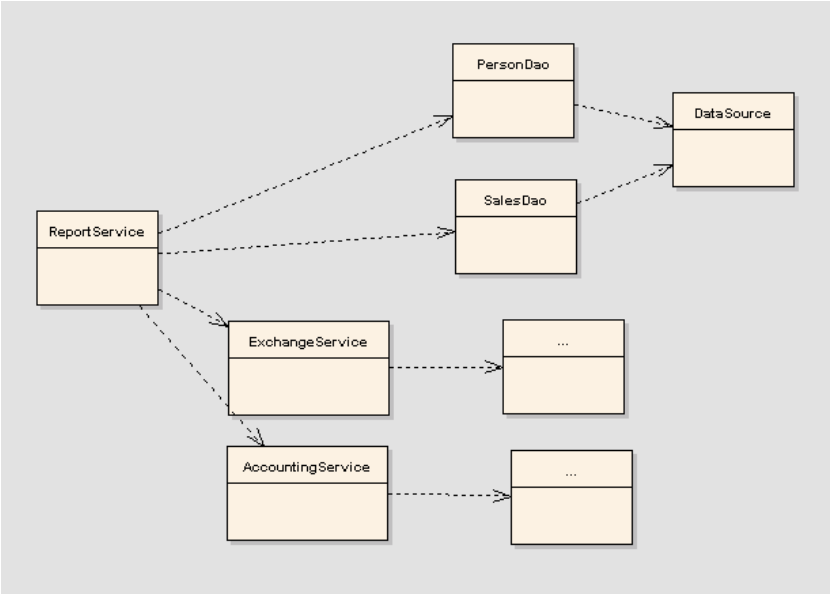
```
@Service
public class BankServiceImpl implements BankService {

    @Override
    public void deposit(Money money, String toAccount) {
        System.out.println("depositing ...");
    }
}
```

Aop näide

```
@Aspect
@Component
public class LoggingAspect {

    private static final Logger log =
                  Logger.getLogger(LoggingAspect.class);

    @Before("execution(* service.*.*(..))")
    public void logBefore(JoinPoint joinPoint)  {

        log.debug("method name: " +
                joinPoint.getSignature().getName());
        log.debug("method arguments : " +
                Arrays.toString(joinPoint.getArgs()));
    }
}
```

Spring


@Component
Client
Spring context
Conf.java


PersonDao
DataSource
SalesDao
ReportService
ExchangeService
...
AccountingService
...

Skoop

- Vaikimisi skoop on “singelton”, ehk üks koopia

```
PersonDao dao1 = ctx.getBean(PersonDao.class);

PersonDao dao2 = ctx.getBean(PersonDao.class);

System.out.println(dao1 == dao2); // true
```

Skoop

```
@Bean
public PersonDao getPersonDao(DataSource dataSource) {
    return new PersonDao(dataSource);
}

@Bean
public DataSource dataSource() {
   DriverManagerDataSource ds = new DriverManagerDataSource();

   ds.setDriverClassName("org.postgresql.Driver");
   ds.setUsername("...");
   ds.setPassword("...");
   ds.setUrl("...");

   return ds;
}
```

Skoobid

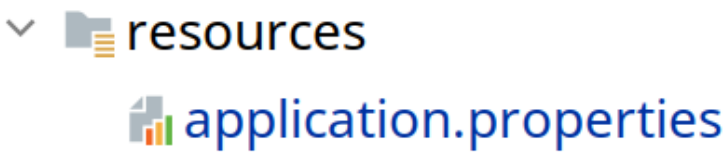
- singleton
- prototype

```
@Component
@Scope("prototype")
public class PersonDao {
```

```
@Component
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public class PersonDao {
```

Välised parameetrid

```
@Configuration
@PropertySource("classpath:/application.properties")
public class Config {

    @Bean
    public DataSource hsqlDataSource(Environment env) {
        ...
        ds.setUrl(env.getProperty("db.url"));

        return ds;
    }
```



application.properties

```
1   db.url=jdbc:postgresql://db.mkalmo.xyz:5432/mkalmo
```

HyperSQL andmebaas

- http://hsqldb.org/
- Mälupõhine andmebaas
- Üks jar fail (1.4 Mb graafiline liides on kaasas)
- Panna teek (jar fail) classpath-ile.

```
implementation group: 'org.hsqldb',
               name: 'hsqldb',
               version: '2.5.1'
```

Hsql süntaks

- Toetab suurt osa SQL:2011 standardist

Hsql süntaks

- PostgreSQL Compatibility
- MySQL Compatibility
- Firebird Compatibility
- Apache Derby Compatibility
- Oracle Compatibility
- DB2 Compatibility
- MS SQLServer and Sybase Compatibility

PostgreSQL Compatibility

```
CREATE TABLE post (
  id SERIAL NOT NULL PRIMARY KEY,
  title VARCHAR(255)
);
```

http://hsqldb.org/doc/2.0/guide/compatibility-chapt.html#coc_compatibility_postgres

Hsql baasi aadress

jdbc:hsqldb:mem:db1

jdbc:hsqldb:mem:db1;sql.syntax_pgs=true

jdbc:hsqldb:file:c:/users/mkalmo/mydb

Skript fail

...

```
ALTER TABLE PUBLIC.ANORDER ALTER COLUMN ID RESTART WITH 4
ALTER SEQUENCE SYSTEM_LOBS.LOB_ID RESTART WITH 1
SET DATABASE DEFAULT INITIAL SCHEMA PUBLIC
GRANT USAGE ON DOMAIN INFORMATION_SCHEMA.YES_OR_NO TO PUBLIC
GRANT USAGE ON DOMAIN INFORMATION_SCHEMA.TIME_STAMP TO PUBLIC
GRANT USAGE ON DOMAIN INFORMATION_SCHEMA.CARDINAL_NUMBER TO PUBLIC
GRANT USAGE ON DOMAIN INFORMATION_SCHEMA.CHARACTER_DATA TO PUBLIC
GRANT USAGE ON DOMAIN INFORMATION_SCHEMA.SQL_IDENTIFIER TO PUBLIC
GRANT DBA TO SA
SET SCHEMA SYSTEM_LOBS
INSERT INTO BLOCKS VALUES(0,2147483647,0)
SET SCHEMA PUBLIC
INSERT INTO PERSON VALUES(1,'Alice')
INSERT INTO PERSON VALUES(2,‘Bob')
INSERT INTO PERSON VALUES(3,'Carol')
```

...

Hsql läbi Spring-i

```
@Configuration
@PropertySource("classpath:/application.properties")
public class HsqlDataSource {

   @Bean
   public DataSource dataSource(Environment env) {
      DriverManagerDataSource ds = new DriverManagerDataSource();

      ds.setDriverClassName("org.hsqldb.jdbcDriver");

      ds.setUrl(env.getProperty("hsql.url"));

      return ds;
   }
```

Projekti 7. osa

- Spring Core raamistiku kasutamine
- HSql andmebaasi kasutamine
- Päringute tegemine kasutades Spring-i võimalusi (JdbcTemplate, SimpleJdbcInsert, ResourceDatabasePopulator)
- Spring-i kontekst luuakse rakenduse käivitamisel ühekordselt.
- Soovitatav on Springi ja baasi osa eraldi valmis teha ja siis rakendusega liita.

hw07 testid

```
public class Hw07 extends AbstractHw {

    private final String baseUrl = "http://localhost:8080/";

    private final String pathToProjectSourceCode = "";
```

Projekti 7. osa

```
@MyController
public class OrderController {

    private OrderRepository repository;

    public OrderController(OrderRepository repository) {
        this.repository = repository;
    }

    @Get("/")
    public String home() {
        return "hw07 home page";
    }
```

Projekti 7. osa

```
@Get("/api/v2/orders")
public List<Order> getAllOrders() {
    return repository.findAll();
}
```

Projekti 7. osa

```
@Post("/api/v2/orders")
public Order createOrder(Order order) {
    return repository.save(order);
}
```

Projekti 7. osa

```
@Get("/api/v2/orders/(\\d+)")
public Order getOrderById(Long id) {
    return repository.findById(id);
}
```

Projekti 7. osa

```
ServletRegistration reg = sce.getServletContext().addServlet(
        "dispatcherServlet", new FrameworkServlet(ctx));

reg.addMapping("/api/v2/*");
```

Projekti 7. osa

```
@Override
public void service(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException {

    String path = request.getRequestURI();

    if ("POST".equals(request.getMethod())) {

        ... ctx.getBeansWithAnnotation(MyController.class)

        // lugeda Json sisend ja teisendada objektiks
        // objekti tüüb refleksiooni abil

        // otsida vastav meetod ja see käivitada
        // andes kaasa Json-ist tehtud objekti

        // teisendada väljund Json-iks ja tagasi saata
```

Cache

```
@Repository
public class TranslationRepository {

    @Override
    @Cacheable("translations")
    public String getTranslation(String key) {
        System.out.println("Fetching translation");
        return dao.getTranslationFor(key);
    }
```

Cache

```
@Repository
public class MemoryMessageRepository {

    @Override
    @Cacheable(value = "message")
    public Message getMessage(String key) {
        System.out.println("Fetching message");
        return messages.get(key);
    }

    @Override
    @CacheEvict(value = "message", key = "#message.key")
    public void save(Message message) {
        messages.put(message.getKey(), message);
    }
```