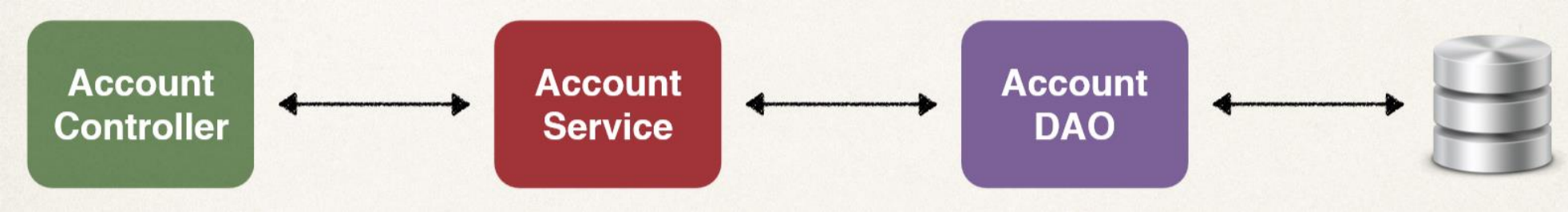
Aspektinis programavimas

Įprastinė sistema

Tarkime turime įprastinę sistemą sudarytą iš Kontrolerio, Serviso ir DAO

Tarkime gavome užduotį jog prieš vykdant DAO metodą mums reikią įrašyti apie tai į LOG failą



Papildomo funkcionalumo pridėjimas

Tarkime mūsų programinis kodas DAO klasėje būtų toks:

```
public void addAccount(Account theAccount, String userId) {
  Session currentSession = sessionFactory.getCurrentSession();
  currentSession.save(theAccount);
}
```

Papildomo funkcionalumo pridėjimas

Tuomet loginimo fun

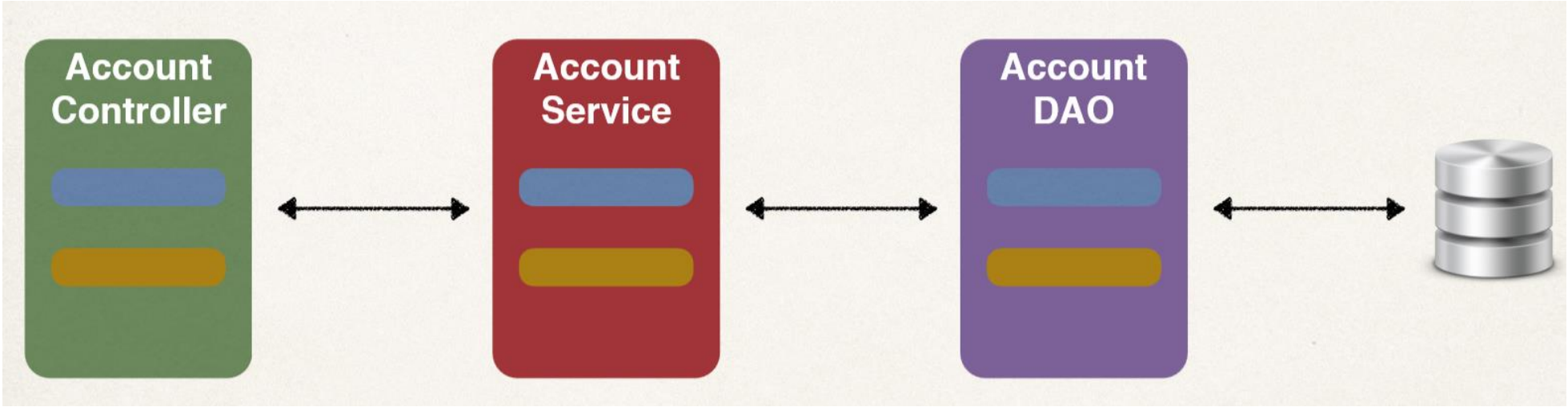
```
public void addAccount(Account theAccount, String userId) {
  //Logo vedimo programinis kodas
  Session currentSession = sessionFactory.getCurrentSession();
  currentSession.save(theAccount);
}
```

Papildomo funkcionalumo pridėjimas

```
public void addAccount(Account theAccount, String userId) {
  //Prisijungimų patikrinimo programinis kodas
  //Logo vedimo programinis kodas
  Session currentSession = sessionFactory.getCurrentSession();
  currentSession.save(theAccount);
}
```

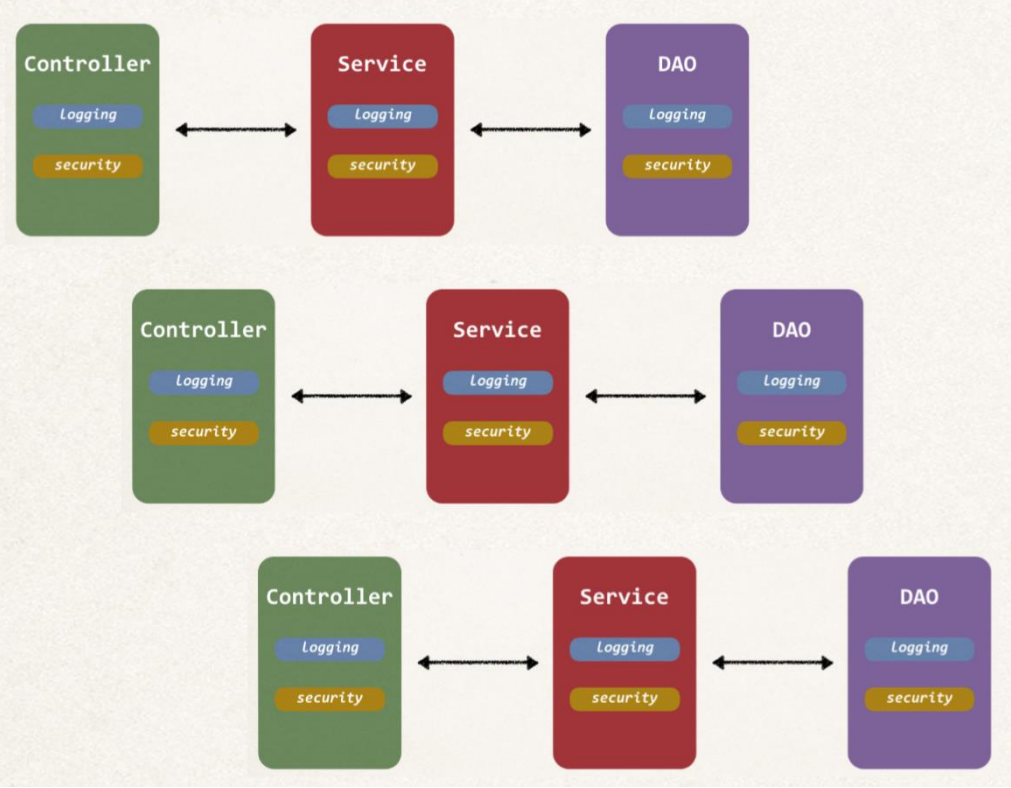
Problemos pridedant funkcionalumą

Įsivaizduokime jog prireikė pridėti logų vedimo programinį kodą visuose sluoksniuose (DAO, Service ir Kontroleryje).



Problemos pridedant funkcionalumą

Laikui bėgant mums gali prireikti pridėti visiems kontroleriams, servisams ir DAO klasėms tokius metodus.



Problemos pridedant funkcionalumą

Jei mes visur surašytumėme tiesiog kodą, turėtumėme tokias problemas:

Pasikeitus Loginimo arba apsaugos kodui jį reikėtų keisti visose vietose.

Jei sukursime statines klases ar kažką panašaus, atsiras supainiotas kodas (vieni metodai kvies kitų klasių metodų kodus).

Kiti sprendimo būdai

Paveldėjimas ....

Visos klasės turėtų paveldėti vieną bendrą klasę kurioje būtų reikiami metodai

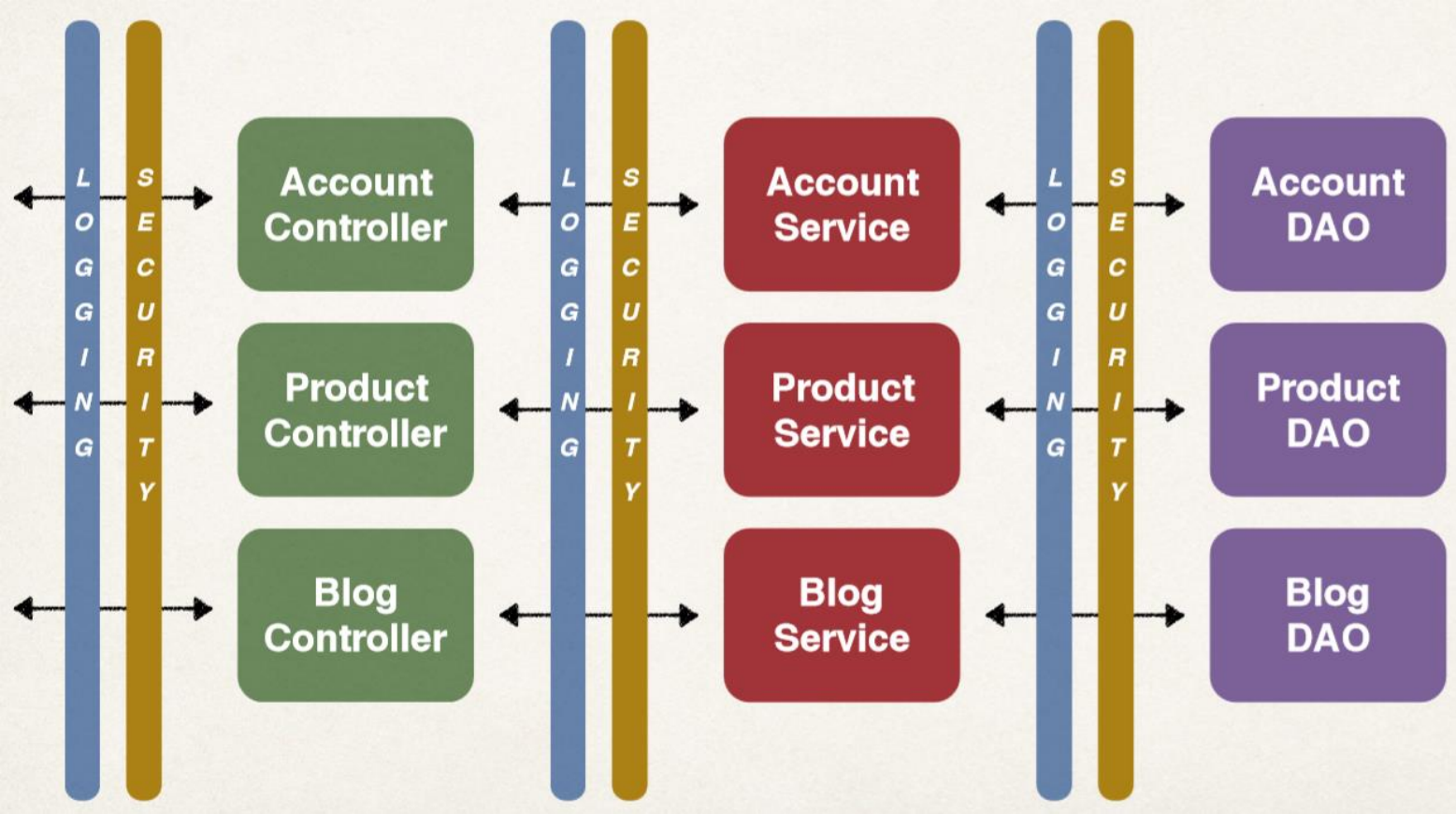
Aspektinis programavimas

Programavimas, kai nustatomi reikalingi programos komponentų poveikio aspektai ir šiais aspektais sistemingai keičiamos programos savybės nekeičiant esamos programos pirminio teksto.

Ypač naudingas, kai programai reikia suteikti naujų savybių, ypač jos veikimą pakreipiant kita linkme. Pavyzdžiui, kai programą reikia papildyti derinimo galimybėmis, rinkti statistiką apie jos atliekamas operacijas ir pan. Tokiais atvejais aspektiniu programavimu galima išvengti tarpkomponentinių ryšių rankinio perprogramavimo.

Aspektinis programavimas praplečia objektinį programavimą ir su juo siejamas. Tačiau gali būti panaudotas ir su kita programavimo paradigma.

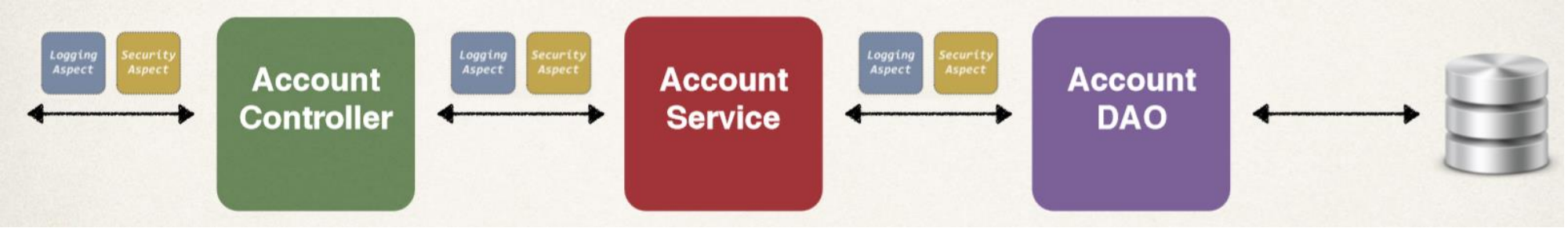
Aspektinio programavimo požiūris



Aspektinis programavimas

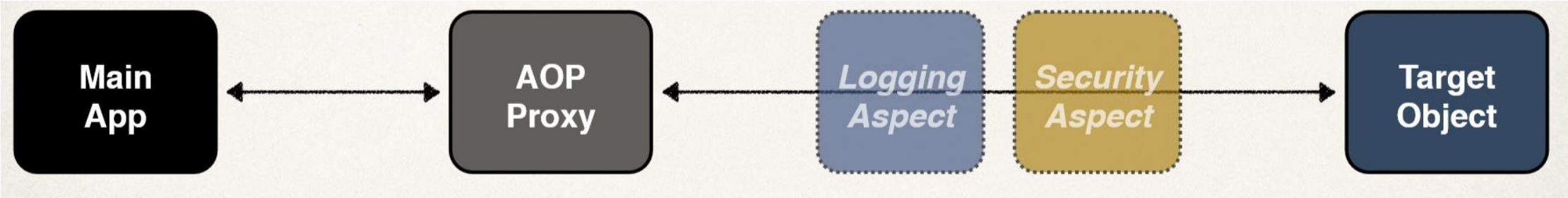
Tie patys aspektai gali būti panaudoti skirtingose vietose

Aspektų taikymas priklauso nuo konfigūracijos



Proxy projektavimo šablonas

Aspektinio programavimo pagrindas Proxy projektavimo šablonas, kuris leidžia įterpti vykdomą kodą tarp iškvietimų.



Privalumai ir trūkumai

PRIVALUMAI

- Reusable modules
- Resolve code tangling
- Resolve code scatter
- Applied selectively based on configuration

TRŪKUMAI

- Too many aspects and app flow is hard to follow
- Minor performance cost for aspect execution (run-time weaving)

Advaice vykdymo būdai

Before advice: run before the method

After finally advice: run after the method (finally)

After returning advice: run after the method (success execution)

After throwing advice: run after method (if exception thrown)

Around advice: run before and after method

Du pagrindiniai AOP framework‘ai

Du dažniausiai naudojami AOP freimwork‘ai JAVA kalbai

AspecJ

Original AOP framework, released in 2001

- www.eclipse.org/aspectj
- Provides complete support for AOP
- Rich support for
- join points: method-level, constructor, field
- code weaving: compile-time, post compile-time and load-time

Užduotis

Psinaudodami Maven parsiųskime AspectJ Weaver biblioteką ir ją instaliuokime

Užduotis

Sukurkime klasę kuriai pritaikysime aspektinį programavimą šią klasę pavadinkime AccountDAO ir joje sukurkime metodą addAccount:

```
@Component
public class AccountDAO {


public void addAccount() {


System.out.println("pridedama vartotojo paskyra");


}
}
```

Konfigūracija

Sukurkime konfigūracinį failą kuriame įjungsime AspectJAutoProxy:

```
@Configuration
@EnableAspectJAutoProxy
@ComponentScan("lt.baltictalents.aspektai")
public class DemoConfig {


}
```

Bandomasis kodas

Pamėginkime pasiimti klasės accountDAO elementą (beans) ir įvykdyti mūsų sukurtą programinį kodą:

```
AnnotationConfigApplicationContext context =
                    new AnnotationConfigApplicationContext(DemoConfig.class);


AccountDAO theAccountDAO = context.getBean("accountDAO", AccountDAO.class);

theAccountDAO.addAccount();


context.close();
```

Užduotis

Sukurkime naują paketą .aspect

Šiame pakete sukurkime naują klasę LoggingAspect kurią naudosime mūsų kodui.

LogingAspect klasė

Klasės LogingAspect programinis kodas:

```
@Aspect
@Component
public class LoggingAspect {
  @Before("execution(public void addAccount())")
  public void beforeAddAccountAdvice() {
    System.out.println("Programinis kodas prieš įterpimą");
  }
}
```

Pointcut filtracijos

Pointcut deklaracijos

https://howtodoinjava.com/spring-aop/aspectj-pointcut-expressions/

https://www.dineshonjava.com/pointcuts-and-wildcard-expressions/

Argumentų paėmimas

Jei mums reiktų paimti argumentus paduodamus į iškviečiamą metodą, tai galėtumėme padaryti tokiu būdu:

```
@Before("lt.baltictalents.aspect.Klase.getSet()")
public void beforeAddAccountAdvice(JoinPoint theJoinPoint) {

  Object[] args = theJoinPoint.getArgs();
  for (Object tempArg : args) {
    System.out.println(tempArg);

    if (tempArg instanceof Account) {
      Account theAccount = (Account) tempArg;
      System.out.println("account name: " + theAccount.getName());
      System.out.println("account level: " + theAccount.getLevel());
    }
  }
}
```

AfterReturning

Metodas bus vykdomas tuomet kai bus pabaigtas stebimas metodas

```
@AfterReturning("execution(* lt.baltictalents.aspect.AccountDAO.findAccounts(..))")
public void afterReturningFindAccountsAdvice() {
System.out.println("Vykdomas isvedimas po metodo @AfterReturning");
}
```

AfterReturning

Norėdami pakeisti grąžintus duomenis iš aspektinės dalies tai galėtumėme atlikti taip:

```
@AfterReturning(
  pointcut="execution(* lt.baltictalents.aspect.AccountDAO.findAccounts(..))",
  returning="result")
public void afterReturningFindAccountsAdvice(
  JoinPoint theJoinPoint, List<Account> result) {


  System.out.println("Vykdomas isvedimas po metodo @AfterReturning");
  //Pakeitus result jis pasikeis visur
}
```