Drugo predavanje

Uvod

U ovom predavanju cilj nam je modelirati domenu. Osim toga, dotaknuti ćemo se i Spring AOP-a, modula koji omogućava aspektno-orijentirano programiranje.

Ovo je definitivno najmanje zahtjevno predavanje :)

Starcraft II

Naša aplikacija će omogućavati će pohranu i prikaz Starcraft II turnira. Ukoliko bude vremena, nadograditi ćemo je mogućnošću da korisnici predviđaju ishod turnira te osvajaju bodove u ovisnosti o uspješnosti.

Zašto Starcraft? Osim što je to najbolja RTS igra svih vremena, postoji profesionalna scena na kojoj se igrači (uglavnom iz Južne Koreje) natječu po turnirima. Evo kako to otprilike izgleda:

Domenski model

U paketu `domain` napraviti ćemo razrede koji predstavljaju ključne entitete:

- Player - predstavlja Starcraft II igrača
- Race - enumeracija sa 4 moguće vrijednosti: Zerg, Protoss, Terran, Random
- Map - enumeracija koja sadrži mape na kojima je moguće odigrati meč
- Game - predstavlja jednu utakmicu u dvoboju
- Match - predstavlja dvoboj dva igrača na turniru
- Round - predstavlja pojedinu rundu turnira, npr. finale, polufinale i sl.
- Tournament - predstavlja cjelokupni turnir

Razrede neću detaljnije raspisivati, sve sam dignuo na GitHub repozitorij.

Application.properties

U `resources` folder projekta dodao sam jednu, naočigled bezazlenu, tekstualnu datoteku - `application.properties`.

No, za Spring Boot ona nije tako bezazlena. Boot će automatski učitati datoteku tog imena ako se ista nalazi u *classpathu*, i koristiti njen sadržaj za konfiguriranje komponenata.

U ovoj fazi, dodati ćemo samo jednostavno konfiguraciju za *logging*:

```
logging.path=./logs/
logging.level.hr.calyx.vjestina2014=INFO
logging.level.org.springframework=ERROR
```

Na taj način reći ćemo Springu da log datoteke sprema u direktorij "`./logs`" i postaviti razine logiranja po pojedinom paketu. Evo mogućih razina:

The six logging levels used by Log are (in order):

1. *trace (the least serious)*
2. *debug*
3. *info*
4. *warn*
5. *error*
6. *fatal (the most serious)*

Te su razine definirane u SLF4J paketu kojeg Spring koristi kao apstrakciju za logiranje (nad kojom je moguće zakačiti različite implementacije, npr. Logback [defaultni] ili Log4j).

Kvalitetno logiranje je važan aspekt svih aplikacija.

Ostale stvari koje se mogu konfigurirati u application.properties datoteci možete pogledati na ovom linku. Više o podršci za logiranje možete pogledati ovdje.

Spring AOP

Druga novost koju uvodimo na ovom predavanju je Spring AOP modul. Za početak, dodajte *dependency* u `build.gradle` i nanovo zavrtite Gradle:

```
dependencies {
   compile("org.springframework.boot:spring-boot-starter-web:1.1.7.RELEASE")
   compile("org.springframework.boot:spring-boot-starter-aop:1.1.7.RELEASE")
}
```

Ako do sada niste bili u kontaktu s aspektno-orijentiranim programiranjem, ukratko možete pročitati na Springovoj dokumentaciji za AOP modul.

Mi ćemo AOP koristiti za logiranje i (kasnije) transakcije. Ako se prvi puta susrećete s AOPom, možda vas uplaši nova terminologija, pa ću izdvojiti 4 koncepta (iz dokumentacije) koja ćemo mi koristiti:

- Aspect: a modularization of a concern that cuts across multiple classes. Transaction management is a good example of a crosscutting concern in enterprise Java applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (the @AspectJ style).
- Join point: a point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution.
- Advice: action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors around the join point.
- Pointcut: a predicate that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by default.

U (novom) paketu, imena `aop`, napraviti ćemo razred `AopLogging` koji će biti `@Aspect` anotiran i koji će definirati sljedeće stvari:

- @Pointcut na sve metode unutar našeg paketa s kontrolerima
- @Around *advice* koji će definirati metodu koju ćemo izvoditi "oko" svake metode koja se pozove unutar `controllers` paketa. Za jednostavno logiranje poziva metode mogli smo koristiti @Before *advice,* ali u našem slučaju želimo bilježiti vrijeme izvođenja svake metode (za buduće optimizacije) pa nam @Before nije dovoljan

```
@Aspect
@Component
public class AopLogging {

   static Logger logger = Logger.getLogger(AopLogging.class);

   @Pointcut("within(hr.calyx.vjestina2014.controllers.*)")
   public void anyController() {};

   @Around("anyController()")
   public Object profile(ProceedingJoinPoint pjp) throws Throwable {
      long start = System.currentTimeMillis();
      Object output = pjp.proceed();
      long elapsedTime = System.currentTimeMillis() - start;
      logger.info("ControllerMethodExecutionTime [method=" + pjp.getSignature().getName() + ",
time=" + elapsedTime + "]");
      return output;
```

```
    }
}
```

Cool zar ne? Na jednom mjestu smo riješili kompletno logiranje. Još coolije će biti kada AOPom riješimo transakcije, koje će se protezati kroz nekoliko razreda u različitim paketima. Ali pustimo nešto uzbuđenja i za kasnije :)

Evo i kako izgleda nekoliko linija u log datoteci. Primjetite da je execution time 0. To će se uskoro promijeniti, kada u priču ubacimo bazu podataka :)

```
2014-10-15 13:38:25.504 INFO 24925 [http-nio-8080-exec-5] --- hr.calyx.vjestina2014.aop.AopLogging :
ControllerMethodExecutionTime [method=getDummyTournament, time=0]
2014-10-15 13:38:25.847 INFO 24925 [http-nio-8080-exec-6] --- hr.calyx.vjestina2014.aop.AopLogging :
ControllerMethodExecutionTime [method=getDummyTournament, time=0]
2014-10-15 13:38:26.146 INFO 24925 [http-nio-8080-exec-7] --- hr.calyx.vjestina2014.aop.AopLogging :
ControllerMethodExecutionTime [method=getDummyTournament, time=0]
2014-10-15 14:40:46.858 INFO 24925 [http-nio-8080-exec-8] --- hr.calyx.vjestina2014.aop.AopLogging :
ControllerMethodExecutionTime [method=getDummyTournament, time=0]
```

.gitignore datoteka

Napomenuti ću i da sam u .gitignore datoteku dodao direktorij “logs/” kako logovi izvođenja ne bi išli na repozitorij. Isto ćemo kasnije napraviti sa konfiguracijskim datotekama, kada će svatko od nas imati neke svoje specifične konfiguracijske parametre.

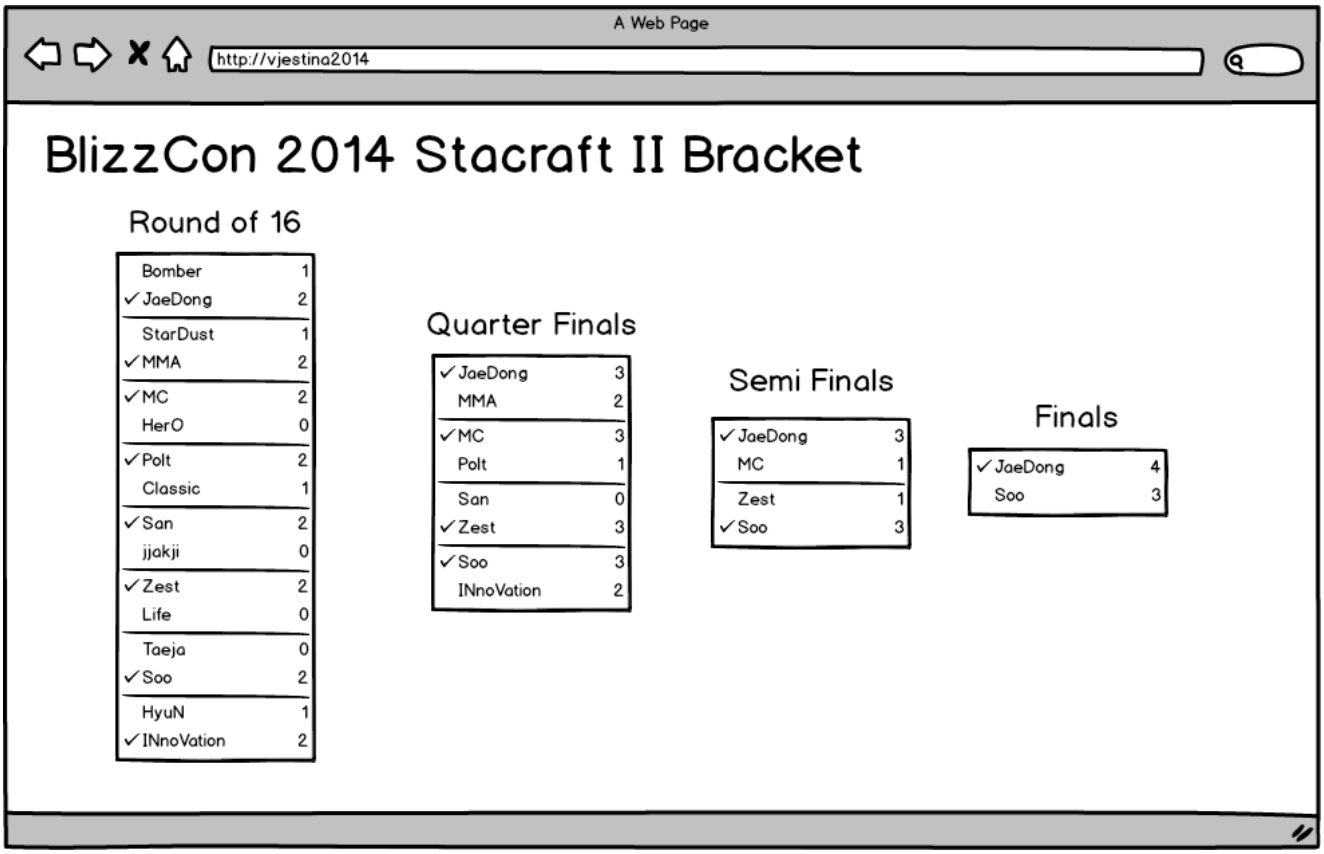
Wireframe zamišljene aplikacije

S obzirom na jednostavnost ovog predavanja, iskoristiti ću vašu nepotrošenu pažnju da definiram zamišljeni izgled konačne aplikacije.

Prikazani ekran predstavlja jedan kompletirani turnir.

Aplikacija će omogućavati definiciju turnira, kao i pregled prethodno odigranih turnira koji se nalaze u bazi.

Bonus (ako stignemo) - mogućnost predviđanja ishoda te bodovanja s obzirom na uspješnost prognoze.


A Web Page
http://vjestina2014
BlizzCon 2014 Stacraft II Bracket
Round of 16
Bomber 1
✓JaeDong 2
StarDust 1
✓MMA 2
✓MC 2
HerO 0
✓Polt 2
Classic 1
✓San 2
jjakji 0
✓Zest 2
Life 0
Taeja 0
✓Soo 2
HyuN 1
✓INnoVation 2
Quarter Finals
✓JaeDong 3
MMA 2
✓MC 3
Polt 1
San 0
✓Zest 3
✓Soo 3
INnoVation 2
Semi Finals
✓JaeDong 3
MC 1
Zest 1
✓Soo 3
Finals
✓JaeDong 4
Soo 3