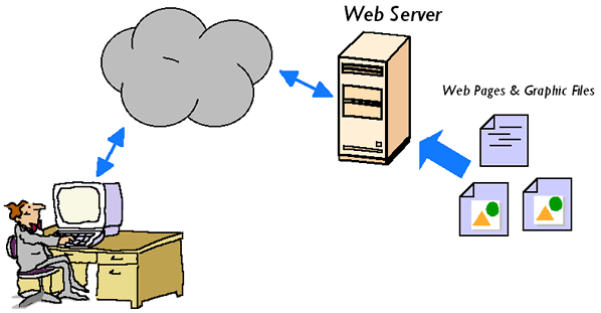
Programvaruteknik DVG302 vt2016

Inlämningsuppgift 5

Servlet med parametrar och Factory

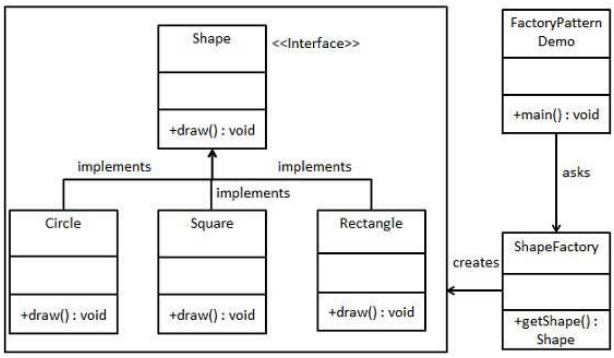
Bakgrund

Denna uppgift är en förlängning av en tidigare inlupp. Som tidigare har vi en servlet som skall utföra något och returnera en Json sträng.



Factory

Ett användbart pattern för denna uppgift skulle kunna vara factory. Ni kommer att behöva olika datakällor så givet en viss inparameter kan ni då från en DataSourceFactory få valfri DataSource-Instans som då klarar av att hämta data från en viss källa. Se exempelvis: http://www.tutorialspoint.com/design_pattern/factory_pattern.htm



Uppgift

Ni skall skapa en servlet som skall sköta om väljandet av två datakällor (*DataSource*) via inskickade parametrar, hämtar och matchar data och returnerar resultatet som *Json* sträng. Hur ni väljer att definiera inparametrarna är upp till er men parametern *pretty* skall också finnas med (som i tidigare uppgift) så att den returnerade Json strängen formateras mer läsligt (återigen som tidigare uppgift).

Dessutom skall ni skapa fler datakällor (minst 2). Minst en ”live” källa, dvs nåt som hämtas från nätet (kan vara nått annat från samma sportlänk som vi tidigare använt eller nått helt annat, som väderdata till exemel) och minst en ”statisk” källa som en slumpvärdeskälla eller en konstantkälla (kan vara bra för testning av systemet senare).

Ni behöver inte ange några andra inparametrar än: `datasource1`, `datasource2` och `pretty`. Alla andra inställningar, typ *date* och *resolution* kan ni hårdkoda på lämpligt sätt, huvudsaken är att ni får fram data som går att matcha. Kom dock ihåg att nämna inställningarna i rapporten.

Ex:
`http://localhost/statistik?datasource1=football&datasource2=temperature`

Skall alltså ge en Json formaterad sträng med resultatet från körningen av de två datakällorna (en enda lång sträng utan whitespace eller return).

```
{"2014-06-01":{"x":2.5,"y":12.6},"2014-04-07":{"x":5.0, osv..
```

```
http:/localhost/statistik?datasource1=football&datasource2=temperature&pret
ty=true
```

Ge en *Json* formaterad sträng med resultatet från körningen av de två datakällorna men som har rader och indenteringar (ungefär som vanlig källkod i era projekt). Detta gör att ni mycket enkelt kan kontrollera att det blir korrekt.

```
{
   "2014-06-01":{
      "x":2.5,
      "y":12.6
   },
   "2014-04-07":{
      "x":5.0,
 osv..
```

TIPS: Använd Rasmus kod ”JasonFormatter” som ligger som bilaga sist i detta dokument.

För övrigt så kan ni för att titta på Json strängar ladda ned en plugin till er webläsare så att det visas på ett mer läsbart sätt. Eller, om ni kör Crome så finns det inbyggt (googla på det).OBS Har man redan pluginen så kan det dölja att eran formatering inte fungerar, kolla upp det.

JUnit:

Att testa funktionaliteten hos en servlet kan vara lite marigt eftersom den existerar i ett ramverk som vi inte har full kontroll över (Tomcat) så denna gång nöjer vi oss med att JUnit testa Factoryklassen för DataSource så att ni ser att den väljer rätt *DataSource*-klass baserat på inskickad parameter. Vill ni ge er på enklare tester av Servleten så kan *mockito* komma till pass (se exempelkoden till förra inluppen här på BB). Problemet är också att kolla upp resultatet från en utskrift (och i princip har vi ju en utskrift från vårt system – Json strängen). Och vi är beroende av vad som ges som resultat från de efterfrågade datakällorna (DataSource). Men inom testning vill vi ju inte vara beroende av andra klasser och källor – det gäller att isolera just den del man vill testa. Här kan man då mocka upp ”fejk”-versioner av sina datakällor och bestämma vad de skall ge för resultat när man frågar dem efter deras data. Då kan man testa att servleten faktiskt skapar rätt datasource. Men det är lite omständigt. Exempelkod ligger på BB om *mockito*. Se även: http://mockito.org/

Ladda ned en jar-fil med *mockito* och inkludera det i ert projekt så kan ni sedan fejka skapandet av datasourceklasserna och har full kontroll över vad de returnerar (som alltså sedan kan jämföras med förväntat returvärde).

Kravet för denna inlupp blir att JUnit testa Factoryklassen, men ni får gärna försöka att testa servleten.

Produktionskod:

Koden blir i form av en servletklass som ni kör på en *Tomcat* server (lokal). Testar gör ni genom att köra det via en Browser (webläsare). Observera att det är bara *Json* strängen som skall skrivas ut i webläsaren (rå eller formaterad enligt ovanstående).

Rapport

Självklart skall ni skriva en rapport, men ni behöver ju inte ordbajsa bara för att få till mycket textvolym, när nu uppgiften är så pass liten. Men alla delar skall finnas med korrekt format och språk. Som resultat går det alldeles utmärkt att visa exempelkörning (skärmdumpar) utöver ren beskrivande text. Formalia enligt tidigare inlämningsuppgift. D.v.s. använd mallen och var noga med referenser och annat.

Kod lämnas endast in som bilaga i rapporten, ingen extra zip-fil. Endast ny produktionskod (servlet, factory, datasource…) och testkod (factory, datakällor, ev. servlet) behöver skickas in.

Bilaga1. Rasmus Östbergs JsonFormatter

```
/**
 * Is used to format strings to a JSON format that is easy to read.
 *
 * @author Rasmus Östberg
 *
 */
public class JsonFormatter {
     /**
      * Formats the given string to a readable JSON format
      *
      * @param unformatedString
      * @return String the formated String.
      */
     public String format(String unformatedString) {
          String base = "";
          int depth = 0;

          for (int i = 0; i < unformatedString.length(); i++) {
               char c = unformatedString.charAt(i);
               if (isBracket(unformatedString.charAt(i))) {
                    if (isleftBracket(c)) {
                         depth++;
                         base += c;
                         base += "\n";
                         base += getTabs(depth);
                    } else {
                         depth--;
                         base += "\n";
                         base += getTabs(depth);
                         base += c;
                    }
               } else if (isComma(c)) {
                    base += c;
                    base += "\n";
                    base += getTabs(depth);
               } else {
                    base += c;
               }
          }
          return base;

     }

     protected boolean isleftBracket(char c) {
          return (c == '[' || c == '{');
     }

     protected boolean isRightBracket(char c) {
          return (c == ']' || c == '}');
     }

     protected boolean isBracket(char c) {
          return isleftBracket(c) || isRightBracket(c);
     }

     protected String getTabs(int tabs) {
          String base = "";
          for (int i = 0; i < tabs; i++) {
               base += "\t";
          }
          return base;
     }

     protected boolean isComma(char c) {
          return c == ',';
     }
}
```