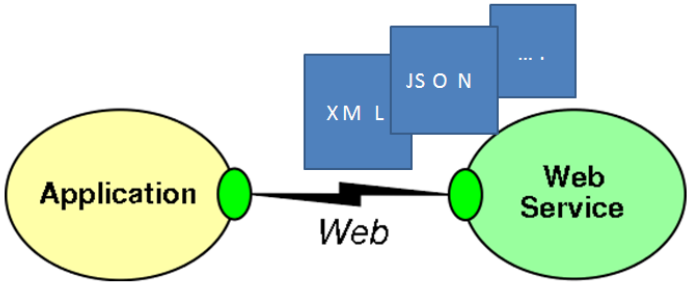
Programvaruteknik – Utveckling och underhåll av mjukvara, DVG302-vt2016

Utvecklingsmodell för inlämningsuppgifter och grund till projektarbetet (forts)

Inlämningsuppgift 2_v1.0

Använda data från webbtjänster



Tanken med denna övning är att du skall kunna skapa kod som anropar web-baserade tjänster och transformera innehållet för det lokala systemets specifika behov

Utförande

Två separata webbtjänster vars responsdata kan konverteras till datum/värde-par skall utnyttjas för att i det hittillsvarande systemet kunna användas som källor för sammanslagning med domänlogiken som konstruerades i inlämningsuppgift 1.

Ni skall alltså kunna läsa data från två olika webtjänster, konvertera detta till det format som vårt interface `DataSource` kräver och ha testat detta.

En klass skall kunna hämta data från en web-tjänst (en klass om implementerar `DataSource`), och en annan klass (en hjälpklass) skall kunna konvertera data(i ett första steg konverterar vi bara till objekt – exempelvis objekt i en Map) till det format som specificeras enligt `DataSource` interfacet. Denna hjälpklass finns redan i workshop-paketet som finns med vid laborationen.

Den ena DataSource klassen skall hämta JSON data från *everysport* och den andra väderdata från någon vädertjänst (exempelvis SMHI). Detta bör vara i annan form, XML eller CSV.

JUnit:

Att testa klasser som jobbar med externt hämtat data kan vara svårt (eftersom man är beroende av den externa källan) så ni får helt enkelt skriva ut den hämtade datasträngen för att ”testa” datahämtningsklassen. Dock kan man tänka sig att ett JUnit test genom att ha en ”extra” metod i `DataSource` klassen som injicerar en fejkad JSON sträng (den har ni ju koll på) och kontrollera att `getData()` levererar rätt svar. Detta är ju ett utomordentligt tillfälle att tillämpa TDD, dvs skriva testet först och koda sedan. Datakonverteringsklassen `DataCollectionBuilder` är ju redan testad i förra laborationen med ”fejkad” indatasträng. Den bör redan ha minst 4-5 stycken olika tester på just det (gärna extremvärden).
Allt implementerat beteende skall täckas av enhetstestning.

Produktionskod:

Ett minimalt klientprogram som skriver ut data från varsin källa och resulterande data (dvs, de sammanslagna data från de två olika källorna).

Som den ena datakällan kan ni använda de målresultat från *everysports* data från allsvenskan 2014 som ni fick via workshopen i början av laborationen och den andra kan vara väderdata (medeltemperatur per dygn). Tanken är att ni skall kolla antalet mål som görs på en viss ort (både hemmalaget och bortalaget) och jämföra det med temperaturen på den orten samma dag för att se om det finns någon korrelation mellan antal mål och temperatur. Alternativ kan man kolla på antal åskådare relaterat till temperaturen. Vill ni glänsa lite så kollar ni olika spelorter och jämför mot temperaturen. Det spelar inte så stor roll vad ni väljer bara det innefattar två riktiga källor där man kan finna data som stämmer i tid och plats. Nu kommer ni alltså att för första gången i produktionskod testa att köra er Matchningsklass på ”riktigt data”.

Redovisning

Denna gång kompletterar vi redovisningen med en formell Rapport. D.v.s. använd mallen (finns på BB) och var noga med referenser och annat. Det behöver inte vara en gigantisk rapport men den skall vara korrekt i sitt formella upplägg och innehålla referenser till aktuella källor (bokavsnitt, web-siter eller andra). Det krävs inte vetenskapliga rapporter eller konferensbidrag men ni bör använda någorlunda ”officiella” källor, inte Sweclockers eller liknande ☺

Lämna även in en zipfil med det exporterad projektet från Eclips.

Lämnas in på BB i samband med redovisningen (samma dag).

Labmaterial

Till denna laboration får ni den kod som användes i workshopen på den första laborationspasset för denna inlämningsuppgift. Det är ett zip-paket med en färdig FootballDataSource och en hjälpklass för att konvertera JSON till en Map-struktur. Det finns även en exempel fil med JSON. Paketet hittar ni på BB under inlämningsuppgift 2.

Nya begrepp

Googla gärna på dessa

- Reading directly from URL (att hämta webdata från ett publict REST API med Java kod)
- REST API
- JSON file
- XML file
- CSV file
- Genson

Extra resurser

Denna klass kan ev. användas som testkälla. Den skapar sinusvärden inlagt i datum från 2012 till och med 2014, den första i varje månad.

```
package [WHATEVER];

import java.time.LocalDate;
import java.util.Map;
import java.util.TreeMap;

public class SineWave implements DataSource {

   @Override
   public String getName() {
      return "sinewave";
   }

   @Override
   public String getUnit() {
      return "undefined";
   }

   @Override
   public Map<LocalDate, Double> getValues() {
      Map<LocalDate, Double> result = new TreeMap<>();
      for (int year = 2012; year < 2015; year++) {
         for (int month = 1; month < 13; month++) {
            LocalDate key = LocalDate.of(year, month, 1);
            result.put(key, Math.sin((key.toEpochDay() - 10957.) / 80.));
         }
      }
      return result;
   }

}
```