Final CAP

Curs 2013-14 (15/I/2014)
Duració: 3 hores.

1. (1 punt) - Què tria `this(Type) && !within(Type)`?
i `within(Type+) && !within(Type)`?
Trien els mateixos *join points*?

Solució:

El primer cas tria tots els *join points* dins de les subclasses de `Type`, el segon tria tots els *join points* dins de les subclasses de `Type`, però a més a més dins de les classes aniuades (*nested classes*) en les subclasses de `Type`.

2. (2.5 punts) - Memoització amb Aspectes. Suposem que tenim definida una classe C on hi ha una funció estàtica `long f (int x) { ... }` tal que costa molt de calcular. Es crida com `C.f(x)`. ***Cal fer un aspecte que guardi una cache de valors de la funció***. És a dir, cada cop que es vulgui avaluar aquesta funció, la crida serà interceptada per mirar si tenim el resultat d'aquesta avaluació per a l'argument `x` ja guardat. En aquest cas es retorna el resultat i no avaluem la funció. Si no el tenim guardat, s'avalua realment `C.f(x)` i es guarda el resultat abans de retornar-lo.

Solució:

```
public aspect Memo {
  private Hashtable memo = new Hashtable();

  pointcut cridesf (int arg) : call(* C.f(..)) && args(arg);

  long around(int arg) : cridesfib(arg) {
     long res;
     Long l = (Long)memo.get(new Long(arg));
     if (l != null) {
       res = l.longValue();
     } else {
       res = proceed(arg);
       memo.put(new Long(arg), new Long(res));
     }
     return res;
  }
}
```

3. (2.5 punts) - Imaginem que tenim una classe, diguem-ne **A**, amb un atribut no privat (protected o public) enter anomenat **grup** (aquest pot ser inicialitzat mitjançant un constructor). Podem considerar que els objectes d'aquesta classe pertanyen a diferents grups en funció del valor d'aquest atribut. Imaginem ara que aquesta classe té un mètode privat, diguem-ne **m()** que retorna una **String**. ***Feu un aspecte que només deixi invocar aquest mètode a objectes que pertanyin al mateix grup***. És a dir, suposem **o1**, **o2** i **o3** són instàncies d'**A**. Si **o1.grup** val 1, **o2.grup** val 2 i **o3.grup** val 1, amb aquest aspecte que us demano **o2** no podria invocar ni **o1.m()** ni **o3.m()**, en canvi **o1** podria invocar **o3.m()** i **o3** podria invocar **o1.m()**.

Solució:

```
public aspect Grups {
   String around() : call(* A.m()) && this(A) && target(A) {
                                     // ^--- això és redundant
                                     //      i no caldria.
       String s;
       A tobj = (A)thisJoinPoint.getThis();
       A ttar = (A)thisJoinPoint.getTarget();
       if (tobj.grup == ttar.grup) {
          s = proceed();
       } else {
          s = "NOPE";
       };
       return s;
    }
}
```

Si haguéssim imposat que l'atribut **grup** fos privat això no funcionaria. N'hi hauria prou, però, afegint **privileged** a la definició de l'aspecte:
privileged public aspect Grups ...

4. (2.5 punts) - Ja vam veure a classe que ECMAScript 5 inclou a **Object** la funció **create(p)** per poder crear objectes amb un prototipus donat **p**. En versions d'ECMAScript anteriors, on **create(p)** no existeix, per fer el mateix hom podia definir aquesta funció:

```
function inherit(p) {
   if (p == null) throw TypeError(); // p no ha de ser null
   var t = typeof p;
   if (t !== "object" && t !== "function") throw TypeError();
   function f() {};
   f.prototype = p;
   return new f();
}
```

Explica amb el màxim detall possible com funciona **inherit(p)**.

Solució:

Després de comprovar que p sigui un objecte adequat com a prototipus d'un altre, el que fa el codi és:

`function f() {};` aquí definim una funció *de mentida*, no cal que faci res
`f.prototype = p;` assignem `p` com a valor de l'atribut `prototype` de `f`
`return new f();`

finalment retornem una *instància* de `f`. És a dir, sabem que si invoquem una funció amb un `new` al davant, l'objecte creat tindrà com a prototipus el mateix objecte referenciat pel valor de l'atribut `prototype` de `f`.

Així doncs, el resultat és un objecte buit (cap propietat) que té com a prototipus a `p`.

5. (1.5 punts) - Suposem que tenim definides dues funcions `A()` i `B()` en Javascript pensades per ser creadores d'objectes (és a dir, per ser cridades amb el `new` al davant: `new A()` i `new B()`). Volem, però, que el prototipus de tots els objectes creats amb `B()` tingui com a prototipus el mateix prototipus que tenen tots els objectes creats amb `A()`. Què cal fer?

Solució:

Senzillament, cal fer dues assignacions *abans* de definir cap mètode ni atribut per a `B.prototype` (perque amb aquestes assignacions estem perdent qualsevol cosa que haguem afegit abans a `B.prototype`):

```
B.prototype = inherit(A.prototype);
B.prototype.constructor = B;
```