

Final CAP

Curs 2014-15 (20/I/2015)

Duració: 2 hores.

Pregunta (1.5 punt): El següent aspecte serveix per fer traces de tots els *join points*:

```
public aspect Tot {
    pointcut p() : call(* *.*(..));
    before() : p() {
        System.out.println(thisJoinPoint);
    }
}
```

però no és correcte. Per què? Com ho arreglaries?

Solució: Crea una recursivitat infinita. Afegint `&& !within(Tot)` a la definició del `pointcut`.

Pregunta (1.5 punt): Considerant només el que hem explicat a classe a l'assignatura de CAP, quin diries que és el canvi més notable en la semàntica de les interfícies (*interface*) Java introduït per AspectJ? Dóna un exemple.

Solució: El fet de poder introduir implementacions *per defecte* dels mètodes que una interfície obliga a implementar. El mateix exemple de les transparències del Tema 3 ens pot servir:

```
public interface Nameable {
    public void setName(String name);
    public String getName();

    static aspect Impl {
        private String Nameable.name;
        public void Nameable.setName(String name) { this.name = name; }
        public void Nameable.getName() { return this.name; }
    }
}
```

Pregunta (1.5 punt): Quina diferència hi ha entre definir un pointcut amb `cflow(...)` o definir-lo amb `cflowbelow(...)`? Il·lustra-ho amb un exemple.

Solució: Això està resolt a les transparències de l'assignatura.

Pregunta (2.5 punts): Vam veure a classe que, si volem simular les classes *tradicionals* en JavaScript, a la Java o Smalltalk, ho podem fer amb certa facilitat. La simulació es complica una mica si a més volíem tenir també herència. Una manera de fer-ho és via prototipus. Si volem fer que el prototipus dels objectes instància de B hereti del prototipus dels objectes instància d'A, cal fer:

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

(on ja sabem què és la funció `inherit`, ho teniu a les transparències). Què passa amb aquesta solució si mètodes *heretats* per les instàncies de B es volen fer servir? Com ho podem arreglar?

Aquest codi us pot servir d'exemple. Volem que la classe B sigui subclasse de la classe A. Fixeu-vos en la sortida:

```
function A() {
```

```

    this.a = 0;
    this.b = 1;
}
A.prototype.retornaA = function() { return this.a }
A.prototype.retornaB = function() { return this.b }

// provem...
var aa = new A();
aa.a = aa.a + 1;
aa.b = aa.b + 1;
console.log(aa.retornaA());
console.log(aa.retornaB());

function B() {
    this.a = 100;
    this.c = 101;
}
B.prototype = inherit(A.prototype);
B.prototype.constructor = B;
B.prototype.retornaC = function() { return this.c }

// provem...
var bb = new B();
console.log(bb.retornaA());
console.log(bb.retornaB());
console.log(bb.retornaC());

```

Solució:

El problema és que els mètodes heretats que depenen d'atributs de la superclasse no funcionaran bé. Això s'arregla fent servir el constructor de la superclasse per afegir els atributs necessaris. En l'exemple, tot plegat s'arreglaria amb una crida a `A.call(this)` dins de `B`:

```

function B() {
    A.call(this);
    this.a = 100;
    this.c = 101;
}

```

Pregunta (3 punts): Volem implementar el patró singleton. Així, com a exemple d'implementació del patró en JavaScript, fem la funció constructora `Universe` amb la intenció que cada cop que faig `new Universe()` ens retorni la mateixa instància. Fem la següent implementació:

```

function Universe() {
    var instance = this;

    this.start_time = 0;
    this.bang = "Big";

    Universe = function () {
        return instance;
    };
}

```

i la provem fent el següent test:

```
Universe.prototype.nothing = true;
var uni = new Universe();
Universe.prototype.everything = true;
var uni2 = new Universe();
console.log(uni.nothing);
console.log(uni2.nothing);
console.log(uni.everything);
console.log(uni2.everything);
console.log(uni.constructor.name);
console.log(uni.constructor === Universe);
```

Executa-ho i respón a les següents preguntes.

- a) Què et sembla el resultat del test? Donaries per bona la implementació d'`Universe` (com a implementació del patró `singleton`) a partir dels resultats d'aquest test?
- b) Si creus que `Universe` està mal implementat, descriu detalladament com funciona i què és el que està malament (fes servir dibuixos com els que vam utilitzar a classe). Si creus que `Universe` està ben implementat, no cal que facis res més.

Solució:

a) el test és incorrecte perquè la darrera línia és `false`, quan no ho hauria de ser. Per tant hi ha alguna cosa que no funciona bé.

b) Cal tenir en compte la redefinició de `Universe` i arreglar el codi per tenir-ho en compte (espero que la vostra resposta sigui *molt* més detallada).