PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Programació

UT 6.2: Interfícies

Joan Pons Tugores

Índex de continguts

Interfícies

És defineixen moltes vegades com a contractes. La interfície defineix una sèrie de mètodes i constants i les classes que l'implementen es comprometen a implementar tots aquests mètodes.

Els mètodes es defineixen posant el selector d'accés, el tipus de tornada, la signatura i les excepcions i acabat amb un punt i coma.

```
public interface MeuInterface{

        // constants

        public static final double E = 2.718282; // base of natural logarithms

        // definicions  dels mètodes

        public void calcula(int i, double x) throws MathematicException;

        public int transforma(String s);

}
```

Per utilitzar una interfície l'ha d'implementar alguna classe, no es poden crear objectes directament de la interfície. Hem d'afegir *implements NomInterface* al final de la declaració de la classe.

```
public class Meva extends Pare implements MeuInterface{
        ....
        public void calcula(int i, double x)  throws MathematicException{
                ...
        }
        public int transforma(String s){
                return ....;
        }
}
```

Una classe pot implementar més d'una interfície. En aquest cas les separarem per , Evidentment aquesta classe haurà d'implementar tots els mètodes de totes les interfícies.

```
public class Meva extends Pare implements MeuInterface, UnAltreInterface{
```

Interfícies com a tipus

En definir un interface estam definint un tipus, com feim en definir una classe, amb la particularitat que no podem crear objectes a partir d'una interfície, sinó que hem de crear objectes d'alguna classe que la implementi. És a dir:

```
MeuInterface n = new MeuInterface(); // Impossible. Error de compilació

Meva m = new Meva(); //Correcte. Meva implementa l'interfície.

MeuInterface p= m; //Polimorfisme
```

Una vegada creat l'objecte, per polimorfisme, podem tractar-lo com si el seu tipus fos la interfície. És a dir, el tipus d'un objecte pot ser:

- el de la seva classe,
- el de qualsevol de les seves superclasses
- el de qualsevol de les interfícies que implementen.

Ho podeu comprovar amb *instanceof*.

Podem tenir un mètode que tengui com a paràmetre un objecte que implementi la interfície, sigui quina sigui la seva classe.

```
public void executa(MeuInterface q){
    //Codi del mètode. Només es podran utilitzar els mètodes i constants de
        //la interfície.
}
Meva p = new Meva();  // Meva implementa MeuInterface.
 executa( p ); //correcte.
```

Utilització d'interfícies com a API

Imaginem que hem dissenyat un sistema de navegació que pot controlar un cotxe, de manera que no necessiti cap conductor. El nostre sistema necessita interactuar amb el cotxe: accelerar, frenar, girar a l'esquerra, ... El problema és que cada fabricant de cotxes funciona d'una manera diferent.

Una solució seria fer una sèrie de classes i que cada fabricant les derivàs, però això limitaria el disseny de les classes de cada fabricant degut a que no disposam d'herència múltiple.

Una solució millor és definir una interfície amb totes les accions, mètodes, que necessita el nostre sistema. El fabricant que el vulgui incloure als seus vehicles n'haurà d'implementar tots els mètodes de forma que funcionin correctament. Renault ho pot implementar de manera molt distinta que Ford, però tots dos podran funcionar amb el nostre sistema, i nosaltres no ens hem de preocupar de com ho han fet.

Els objectes del nostre sistema rebran com a arguments objectes que implementin les interfícies que hem definit i així podran interactuar amb ells.

En aquest cas podríem veure la interfície com una API.

Modificació d'interfícies

S'ha d'anar molt alerta en modificar una interfície canviant la definició dels seus mètodes (afegint paràmetres, canviant el tipus de tornada, ...), afegint mètodes, ...

En fer algun d'aquests canvis provocarem que automàticament totes les classes que l'implementin donin error de compilació, ja que han d'implementar tots els mètodes de la interfície tal qual estan definits.

Per això és importat tenir ben dissenyades les interfícies abans, sobre tot, de distribuir-les.

Herència d'interfícies

Per les interfícies també existeix el concepte d'herència, i en aquest cas sí que admeten herència múltiple. Per exemple puc tenir

```
public interface MeuInterface extends Interficie1, Interficie2{

        // constants

        public static final double E = 2.718282; // base of natural logarithms

        // signatures  dels mètodes

        public void calcula(int i, double x);

        public int transforma(String s);

}
```

MeuInterface heretarà tots els mètodes i constants definits per *Interfície1* més els definits per *Interfície2*. Per tant una classe que vulgui implementar *MeuInterface* haurà d'implementar també tots aquests mètodes.

Crear una interfície derivada pot ser una bona manera de modificar una interfície sense provocar una allau de queixes i errors de compilació.