Interfícies

De vegades és útil definir què ha de fer una classe però no com ha de fer-ho. Ja hem vist un exemple d'això en forma de mètodes abstractes. Un mètode abstracte defineix la interfície (aparença externa?) del mètode però no defineix la seva implementació.

Les classes i els mètodes abstractes són útils i es pot portar aquest concepte més enllà: a Java es pot separar la interfície de la implementació d'una classe fent servir la paraula reservada **interface**:

```
public interface Amable {
	public void saluda();
}
```

Aquest és un exemple de interfície. **Són semblants a les classes abstractes amb l'excepció que han de tenir tots els mètodes sense implementar**, és a dir, abstractes. Quan es defineix una interfície, ja se sap que tots els mètodes són abstractes, però es pot posar si es vol a la signatura del mètode (no s'acostuma a fer)

```
public interface Amable {
	public abstract void saluda();
}
```

Els mètodes d'una interfície han de ser tots públics, en canvi, les interfícies poden ser públiques o amb visibilitat default (tan sols visibles al mateix package).

Les interfícies poden tenir variables, però tan sols variables públiques, estàtiques i finals, és a dir, constants. Si no es posa res, es sobreentén que són públiques, estàtiques i finals. Per exemple, aquesta definició:

```
public interface Amable {
	String MISSATGE = "Hola";

	public abstract void saluda();
}
```

i aquesta són equivalents:

```
public interface Amable {
	public static final String MISSATGE = "Hola";

	public abstract void saluda();
}
```

Implementant interfícies

Un cop s'ha definit una interfície, podem fer que una o més classes **implementin** aquesta interfície. Per això es necessari la paraula reservada **implements**.

Per exemple, si tenim la interfície Amable:

```
package exemple3;

public interface Amable {

	public abstract void saluda();
}
```

Podem definir la classe Recepcionista que implementarà aquesta interfície:

```
package exemple3;

public class Recepcionista implements Amable {

	@Override
	public void saluda() {
		System.out.println("Hooola");
	}
}
```

El fet d'implementar una interfície, igual que amb les classes abstractes, obliga a la classe a implementar els mètodes definits en ella.

El principal avantatge de les interfícies és que una classe pot implementar més d'una interfície. A més, una classe pot estendre a una altre classe i implementar una, dos o les interfícies que sigui necessari. Per exemple:

```
public class Cotxe extends Vehicle implements Mobible, ContenidorPersones {
...
}
```

Les interfícies a implementar han d'anar al final de la declaració de classe i separades per comes.

Variables a les interfícies

Les variables d'una interfície són necessàriament públiques, estàtiques i finals. Això és una limitació, però en certes situacions pot ser útil. Per exemple, de vegades un programa necessita definir un conjunt de constats. Una manera de fer-ho és mitjançant una interfície:

```
package exemple4;

public interface Constants {
	int MIN = 0;
	int MAX = 100;
	String ERROR_MSG = "Error!";
}
```

Per a fer servir aquestes variables hi ha dues opcions. La primera és implementant aquesta interfície:

```
package exemple4;

public class TestConstants implements Constants {

	public static void main(String[] args) {
		int[] array = new int[MAX];

		//...
	}
}
```

És una interfície buida de mètodes! No suposa cap esforç implementar-la i permet fer servir les constants definides a la interfície directament, com si s'haguessin definit a la mateixa classe.

L'altre opció és no implementar-la i fer servir les constants definides de la mateixa manera que les faríem servir si són definides a una classe normal:

```
public class TestConstants {

	public static void main(String[] args) {
		int[] array = new int[Constants.MAX];

		//...
	}
}
```

Les interfícies es poden estendre

Una interfície pot estendre a una altre interfície (o a més d'una!):

```
package exemple5;

public interface A {
	public int metode1();
}
```

```
package exemple5;

public interface B extends A {
	public int metode2();
	public int metode3();
}
```

```
package exemple5;

//ha d'implementar els mètodes d'A i B!
public class C implements B {

	public int metode1() {
		return 0;
	}

	public int metode2() {
		return 0;
	}

	public int metode3() {
		return 0;
	}
}
```

Exemple d'aplicació

La classe **java.lang.Thread** serveix per a representar un fil d'execució d'un programa.

Aquesta classe també té mètodes estàtics per a gestionar el fil que està executant el programa. Per exemple, podem adormir (aturar) el programa un número determinat de mili-segons cridant al mètode *sleep*:

```
package exemple6;

public class SleepThreadTest {

	public static void main(String[] args) {
		for(int i=0;i<10;++i) {
			System.out.println("Han pasat " + i + " segon(s)");
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				System.err.println("Error: " + e.toString());
			}
		}
	}
}
```

Una de les maneres de definir una tasca a ser executada per un Thread és proporcionant la tasca a fer en forma d'objecte que implementa l'interfície **java.lang.Runnable.**

```
package exemple6;

public class Task1 implements Runnable {

	public void run() {
		for(int i=0;i<10;++i) {
			System.out.println("Han pasat " + i + " segon(s)");
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				System.err.println("Error: " + e.toString());
			}
		}
	}
}
```

I li paso la tasca en forma d'objecte al constructor del Thread:

```
package exemple6;

public class TaskThreadTest {

	public static void main(String[] args) {

		Task1 task1 = new Task1();
		Thread thread1 = new Thread(task1);
		thread1.start();

	}
}
```

Aquest és un exemple de fer servir les interfícies com a mecanisme per a que una llibreria o *framework* es pugui fer servir d'una manera fàcil i sense que afecti al model jeràrquic de classes de l'usuari.

Llicència

Copyright (C) Alfonso da Silva (alfonsodasilva@gmail.com)
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is in http://www.gnu.org/licenses/fdl.txt