CFGS Desenvolupament d'Aplicacions Multiplataforma (DAM)

CFGS Desenvolupament d'Aplicacions Web (DAW)

Mòdul 5 – Entorns de Desenvolupament.
UF2 – Optimització de programari

Exemples de refacció

Índex

Índex de contingut

Codi inicial

Tenim el següent codi, que demana el preu net de 4 productes i escriu el preu que resulta d'aplicar l'IVA (21%):

```
// formatat per Eclipse

package refaccio;
import java.util.Scanner;

public class PerFerRefaccio {


	public static void main(String[] args) {
		Scanner entrada=new Scanner(System.in);

		float sumaSenseIVA=0,sumaAmbIVA=0;

		System.out.println("Entra els 4 imports");
		System.out.println();


		for(int i=1;i<=4;i++){
			System.out.println("Entra l'import nro. "+i+" ");

			float preu=entrada.nextFloat();

			sumaSenseIVA+=preu;

			System.out.println("Amb IVA es "+(preu+preu*21/100));
			System.out.println();
			sumaAmbIVA+=(preu+preu*21/100);
		}

		entrada.close();
		System.out.println();
		System.out.println("El total dels 4 imports es "+
					sumaSenseIVA+"; amb IVA "+sumaAmbIVA);
	}

}
```

En els següents apartats aplicarem diferents tècniques de refacció per millorar aquest codi amb la finalitat de fer-lo més eficient i/o fàcilment legible i/o fàcilment mantenible (modificable).

Extreure una variable local

El primer que crida l'anteció és que el càlcul `(preu+preu*21/100)` es repeteix. Això pot evitar-se posant el seu resultat en una variable, per exemple, *preuAmbIVA*. Es destaca amb fons groc la part modificada.

```
// formatat per Eclipse

package refaccio;
import java.util.Scanner;

public class ExtraccioVariable {


	public static void main(String[] args) {
		Scanner entrada=new Scanner(System.in);

		float sumaSenseIVA=0,sumaAmbIVA=0;

		System.out.println("Entra els 4 imports");
		System.out.println();

		for(int i=1;i<=4;i++){
			System.out.println("Entra l'import nro. "+i+" ");

			float preu=entrada.nextFloat();
			float preuAmbIVA=(preu+preu*21/100);
			sumaSenseIVA+=preu;

			System.out.println("Amb IVA es "+preuAmbIVA);
			System.out.println();
			sumaAmbIVA+=preuAmbIVA;
		}

		entrada.close();
		System.out.println();
		System.out.println("El total dels 4 imports es "+
					sumaSenseIVA+"; amb IVA "+sumaAmbIVA);
	}

}
```

Extreure una constant

Què passaria si canviessin les especificacions del programa i, en lloc d'haver d'entrar 4 valors, calgués fer el mateix tractament sempre amb 3 valors? I amb 8?

El que passaria és que cada cop que hi hagués un canvi d'aquests, caldria repassar manualment el programa per canviar els quatres pel nou número. En aquest procés és molt fàcil que es produeixin errors (descuidar-nos de canviar algun 4 o canviar algun 4 que indica alguna cosa diferent al nombre de valors). Això es pot solucionar amb la tècnica de refacció "extreure una constant". Suposem sempre que aquests canvis són molt poc freqüents (sinó, caldria extreure novament una variable).

Si fem els canvis sobre el codi anterior, el resultat podria ser aquest (novamanent, s'assenyalen els canvis amb fons groc):

```
// formatat per Eclipse

package refaccio;
import java.util.Scanner;

public class ExtraccioConstant {


	public static void main(String[] args) {
		Scanner entrada=new Scanner(System.in);

		float sumaSenseIVA=0,sumaAmbIVA=0;
		final int nImports=4;

		System.out.println("Entra els "+nImports+" imports");
		System.out.println();

		for(int i=1;i<=nImports;i++){
			System.out.println("Entra l'import nro. "+i+" ");

			float preu=entrada.nextFloat();
			float preuAmbIVA=(preu+preu*21/100);
			sumaSenseIVA+=preu;

			System.out.println("Amb IVA es "+preuAmbIVA);
			System.out.println();
			sumaAmbIVA+=preuAmbIVA;
		}

		entrada.close();
		System.out.println();
		System.out.println("El total dels "+nImports+" imports es "
					+sumaSenseIVA+"; amb IVA "+sumaAmbIVA);
	}

}
```

Amb aquest canvi, si ens modifiquen el nombre d'imports, només cal canviar el valor de la constant.

Extreure un mètode

Per veure millor els avantatges de la refacció *extreure un mètode* canviarem lleugremanet l'exemple. Ara es tracta de demanar 4 imports i escriure, per a cadascun, el resultat d'aplicar-li un descompte petit (per exemple, per minoristes) i un import gran (per exemple, per majoristes). Teniu la situació inicial a continuació. Veureu que ja s'hi han fet les refaccions anteriors (extreure variables i constants), però que, tot i així, encara hi ha un càlcul que es repeteix, tot i que amb diferents valors.

```
// formatat per Eclipse

package refaccio;
import java.util.Scanner;

public class PerExtreureMetode {


	public static void main(String[] args) {
		Scanner entrada=new Scanner(System.in);

		final int nImports=4;
		final float descomptePetit=5, descompteGran=10;

		System.out.println("Entra els "+nImports+" imports");
		System.out.println();

		for(int i=1;i<=nImports;i++){
			System.out.println("Entra l'import nro. "+i+" ");

			float preu=entrada.nextFloat();
			float preuAmbDescomptePetit=(preu-preu*descomptePetit/100);
			float preuAmbDescompteGran=(preu-preu*descompteGran/100);

			System.out.println("Amb descomptePetit:"
					+preuAmbDescomptePetit+"; amb descompte gran:"
							+preuAmbDescompteGran);

		}

		entrada.close();

		System.out.println();
		System.out.println("Final dels calculs");
	}

}
```

En aquest cas, extraurem com a mètode el càlcul de l'import amb el descompte aplicat.
Guanyarem el següent:

- Legibilitat: al nom del mètode posem què fa el càlcul; en aquest cas podriem haver-ho fet amb un comentari, ja que només ocupa una línia, però en molts casos això no és així. D'altra banda, si ocupés més d'una línia costaria al programador que ho llegís veure on acaben les instruccions del càlcul.
- Modificabilitat: si la manera de calcular el descompte canviés, amb aquest canvi només caldria modificar el codi a un lloc; això fa molt més difícil que ens equivoquem.

Fixeu-vos que el primer avantatge l'hauriem tingut també amb l'exemple anterior, però no el segon.

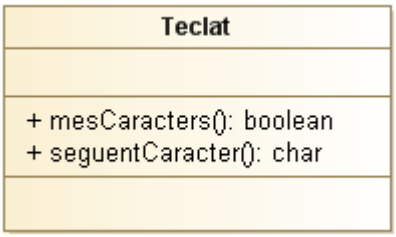
El resultat de fer el canvi és el següent (també estan destacats amb fons groc els canvis):

```
// formatat per Eclipse

package refaccio;
import java.util.Scanner;

public class MetodeExtret {


	public static void main(String[] args) {
		Scanner entrada=new Scanner(System.in);

		final int nImports=4;
		final float descomptePetit=5, descompteGran=10;

	    System.out.println("Entra els "+nImports+" imports");
	    System.out.println();

		for(int i=1;i<=nImports;i++){
			System.out.println("Entra l'import nro. "+i+" ");

			float preu=entrada.nextFloat();
			float preuAmbDescomptePetit=calculaDescompte(descomptePetit,
			                                                    preu);
			float preuAmbDescompteGran=calculaDescompte(descompteGran,
			                                                    preu);

			System.out.println("Amb descomptePetit:"+
			                        preuAmbDescomptePetit+"; amb descompte gran "+
			                                                preuAmbDescompteGran);

		}

		entrada.close();

		System.out.println();
		System.out.println("Final dels calculs");
	}

	private static float calculaDescompte(final float descomptePetit, float preu)
	{
		return preu-preu*descomptePetit/100;
	}

}
```

Extreure una interfície

Per il·lustrar la utilitat d'aquest tipus de refacció també canviarem d'exemple.
Suposem que és un dispositiu que proporciona caràcters seqüencialmente (com per exemple un teclat) i que requereix instruccions molt especifiques per obtenir aquests caràcters. Una bona pràctica consisteix en separar la implementació de l'accés a aquest dispositiu de la resta del programa.

Això pot fer-se amb una classe, com, per exemple, la següent (s'ometen les dades perquè depenen de la codificació i no la farem):



Un programa que hi accedís podria tenir instruccions com aquestes:

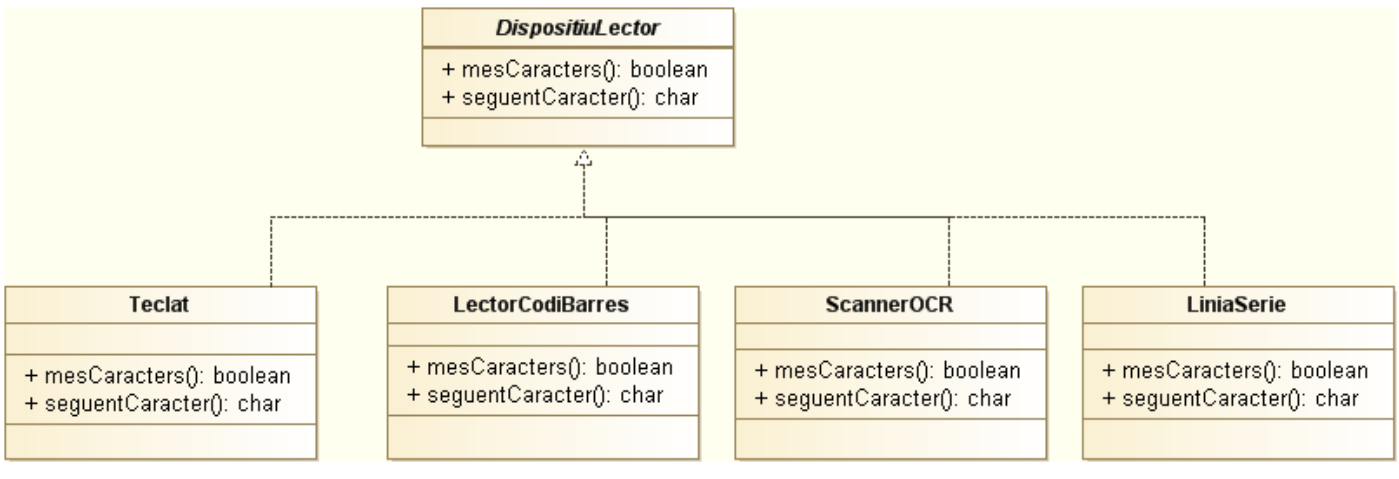
```
Teclat teclat = new Teclat();


if(teclat.mesCaracters()) {
      System.out.println(teclat.seguentCaracter());
}
```

Què succeeix si:

- Ara necessitem treballar amb més dipositius que també proporcionen caràcters seqüèncialment (per exemple, lector de codi de barres, scannerOCR, liniaSerie…) i que requereixen instruccions molt diferents
- Volem reaprofitar els programes ja realitzats per al teclat.

Per a cada tipus de dispositiu caldrà fer una classe que tingui els mateixos mètodes que l'anterior. D'aquesta manera, als programes només caldrà modificar la instrucció on es crea l'objecte per una que rebi l'objecte com a paràmetre. També seria convenient canviar el nom de la variable per un de més genèric (per exemple, dispositiuLector). L'únic problema que ens queda per resoldre és com declarem aquesta variable. Si la declarem com a Teclat, no acceptarà que li assignem qualsevol altre tipus d'objecte. El mateix passaria si la declarem com a LectorCodiBarres, ScannerOCR, etc. Solució: declarar una interfície de Java (l'anomenarem *DispositiuLector*) que tingui tots aquests mètodes i fer que la resta de classes la implementin. Així li diem al compilador que totes aquestes classes sobreescriuen els seus mètodes i, per tant, poden utilitzar-se de manera uniforme i, a efectes pràctics, poden considerar-se totes del tipus *DispositiuLector*.

Gràficament quedaria com es mostra a la següent figura (a les classes s'ometen les dades perquè depenen



```
// formatat per Modelio

public interface DispositiuLector {

   boolean mesCaracters();

   char seguentCaracter();

}
```

I el de les classes, seria semblant a aquest:

```
// formatat per Modelio

public class Teclat implements DispositiuLector {

// dades necessàries

   public boolean mesCaracters() {
// implementació del mètode mesCaracters
   }

   public char seguentCaracter() {
// implementació del mètode seguentCaracter
   }
}
```

```
// formatat per Modelio

public class LectorCodiBarres implements DispositiuLector {

// dades necessàries

   public boolean mesCaracters() {
// implementació del mètode mesCaracters
   }

   public char seguentCaracter() {
// implementació del mètode seguentCaracter
   }
}
```

```
// formatat per Modelio

public class ScannerOCR implements DispositiuLector {

// dades necessàries

   public boolean mesCaracters() {
// implementació del mètode mesCaracters
   }

   public char seguentCaracter() {
// implementació del mètode seguentCaracter
   }
}
```

```
// formatat per Modelio

public class LiniaSerie implements DispositiuLector {

// dades necessàries

   public boolean mesCaracters() {
// implementació del mètode mesCaracters
   }

   public char seguentCaracter() {
// implementació del mètode seguentCaracter
   }
}
```

L'exemple anterior de codi quedaria així (per exemple):

```
void funcio(DispositiuLector dispositiuLector){

//.... codi


if(dispositiuLector.mesCaracters()) {
      System.out.println(dispositiuLector.seguentCaracter());
}
```