

ELO329 - Diseño y Programación Orientados a Objetos

Expresiones Lambda

Agustín González

Patricio Olivares

Expresiones Lambda

- Cuando la implementación de una clases anónima es simple, la sintaxis de las clases anónimas puede ser excesiva y poco clara.
- En estos casos se busca pasar cierta funcionalidad como un argumento de otro método.
- Las **Expresiones Lambda** permiten tratar funcionalidad como un argumento, o código como datos.
- Las expresiones lambda nos permiten expresar instancias de una clase **de método único** de manera más compacta.

Recordemos la clase BankAccount

```
class BankAccount {  
    public BankAccount(double initialBalance) {  
        balance = initialBalance;  
    }  
  
    public void start(final double rate) {  
        // Clase anónima  
        ActionListener adder = new ActionListener() { // adder es la única instancia  
            public void actionPerformed(ActionEvent event) { // implementación  
                double interest = balance * rate / 100;  
                balance += interest;  
                NumberFormat formatter = NumberFormat.getCurrencyInstance();  
                System.out.println("balance=" + formatter.format(balance));  
            }  
        };  
        Timer t = new Timer(1000, adder);  
        t.start();  
    }  
    private double balance;  
}
```

Observaciones: método start

```
public void start(final double rate) {  
    ActionListener adder = new ActionListener() { // adder es la única instancia  
        public void actionPerformed(ActionEvent event) { // implementación  
            double interest = balance * rate / 100;  
            balance += interest;  
            NumberFormat formatter = NumberFormat.getCurrencyInstance();  
            System.out.println("balance=" + formatter.format(balance));  
        }  
    };  
    Timer t = new Timer(1000, adder);  
    t.start();  
}
```

- Observaciones:
 - `adder` es creado solo para pasarlo como argumento.
 - Lo que se desea es entregar al timer `t` el código que debe ser invocado periódicamente.
 - Notar que `ActionListener` es una interfaz con solo un método.

Observaciones: método start

- En este caso podríamos omitir la creación de `adder`. El método queda así:

```
public void start(final double rate) {  
    Timer t = new Timer(1000, new ActionListener() {  
        public void actionPerformed(ActionEvent event) {  
            double interest = balance * rate / 100;  
            balance += interest;  
            NumberFormat formatter = NumberFormat.getCurrencyInstance();  
            System.out.println("balance=" + formatter.format(balance));  
        }  
    });  
    t.start();  
}
```

Observaciones: método start

- Cuando se trata de un solo método y dado que es aquel de interfaz `ActionListener`, se puede simplificar y omitirlo. Lo mismo con el nombre de la interfaz. **Así se llega a la versión usando un expresión lambda:**

```
public void start(final double rate) {  
    Timer t = new Timer(1000, event -> {  
        double interest = balance * rate / 100;  
        balance += interest;  
        NumberFormat formatter = NumberFormat.getCurrencyInstance();  
        System.out.println("balance=" + formatter.format(balance));  
    });  
    t.start();  
}
```

Sintaxis General de las Expresiones Lambda

- `(a, b, ..., c) -> { sentencia 1; sentencia 2; ...; }`
- `a`, `b`, ..., `c` son los parámetros del método. Si es solo un parámetro, podemos omitir los paréntesis `()`.
- Le sigue la flecha `->`
- Le sigue una sentencia única o un bloque de sentencias.
- En cada expresión lambda se crea un objeto.
- Equivale a un `new Alguna_Interfaz(){...}`, donde `Alguna_Interfaz` tiene un único método con parámetros `(a, b, ..., c)` con implementación `{sentencia 1; sentencia 2; ...}`
- Veamos otro ejemplo en otro contexto →

Creación de instancia usando expresión lambda

- Ver `Calculator.java`

```
public class Calculator {  
    interface IntegerMath { // interfaz anidada  
        int operation(int a, int b);  
    }  
    public int operateBinary(int a, int b, IntegerMath op) {  
        return op.operation(a, b);  
    }  
    public static void main(String... args) {  
        Calculator myApp = new Calculator();  
        IntegerMath addition = (a, b) -> a + b;  
        IntegerMath subtraction = (a, b) -> a - b;  
        System.out.println("40 + 2 = " + myApp.operateBinary(40, 2, addition));  
        System.out.println("20 - 10 = " + myApp.operateBinary(20, 10, subtraction));  
    }  
}
```