Apuntes de ingeniería del software

Java Closures

José Juan Hernández Cabrera
Escuela Universitaria de Ingeniería Informática. Universidad de Las Palmas de Gran Canaria
Enero 2013

¿Qué son?

Un **closure** nos permite pasar métodos como punteros a otras clases o métodos. Los closures permiten incluso encapsular el contexto (estados almacenados en variables) que se pasan junto con el puntero al método.

En algunos lenguajes de programación también se conocen como **Expresiones Lambda o Métodos Anónimos**

Mediante los closures, se reduce la sobrecarga de codificación al poder crear instancias de closures de forma más simple.

Mecanismos en Java

Hasta la versión 7 de Java no existe la posibilidad de crear closures aunque conceptualmente habitualmente se especifican.

```
public interface Function {
  public double apply(double value);
}

public class SquareFunction implements Function {
  @Override
  public double apply(double value) {
    return value * value;
  }
}
```

The Lambda Project

`http://openjdk.java.net/projects/lambda/`

El proyecto Lambda va a integrarse en la nueva versión de Java SE 1.8.

Esta versión se liberará en 2013

Expresiones lambda

```
(String a, String b) -> a.compareTo(b)

(int x, int y) -> x + y

() -> 42

(String s) -> { System.out.println(s); }
```

Interfaces funcionales

Son interfaces que tienen un sólo método.

Estas interfaces representan un contrato funcional.

A las interfaces funcionales también se las conoce como SAM (Single Abstract Method)

Interfaces funcionales

```
public interface FileFilter {
   public boolean accept(File file);
   }
}
```

Interfaces funcionales

Se pueden crear instancias con clases anónimas.

```
FileFilter filter = new FileFilter() {
  @Override
  public boolean accept(File file) {
    return file.getName().endsWith(".xml");
  }
}
```

Interfaces funcionales

La nueva forma de crear una instancia con expresiones lambda.

```
FileFilter filter =
   (File file) -> file.getName().endsWith(".xml");
```

Código inteligente

La filosofía de Java se basa en eliminar redundancias para facilitar la lectura del código

Antes de Java SE 7

```
Map<String,Integer> m1 = new HashMap<String Integer>();
```

A partir de Java SE 7

```
Map<String,Integer> m1 = new HashMap<>();
```

Interfaces funcionales

Como el compilador conoce los tipos de la interface funcional, se puede hacer aún más compacta.

```
FileFilter filter =
  file -> file.getName().endsWith(".xml");
```

Otro ejemplo

```
public interface Comparator<T> {
  public int compare(T o1, T o2);
}

Comparator<String> c =
  (s1, s2) -> s1.compareToIgnoreCase(s2);
```

Otro ejemplo

```
public interface Callable<T> {
  public T call();
}

Callable<String> callable = ()-> "done";

Callable<String> callable = ()->{return "done”;};
```

Otro ejemplo

```
public interface Block<T> {
  public void apply(T o);
}

Block<Account> block =
  (a) -> { if (a.balance() < min) a.alert(); };
```

Ejercicio

Implementar la clase SquareFunction como Closure con expresiones lambda.

```
public interface Function {
  public double apply(double value);
}


public class SquareFunction implements Function {
  @Override
  public double apply(double value) {
    return value * value;
  }
}
```

Ejercicio resuelto

```
public interface Function {
  public double apply(double value);
}

Function function = value -> value * value;
```

Referencias a métodos

```
public class ComparatorProvider {
  public int compareByName(Person p1, Person p2) {
    return p1.getAge() - p2.getAge();
  }
 public int compareByMoney(Person p1, Person p2) {
    return p1.getMoney() - p2.getMoney();
  }
}

ComparatorProvider provider = new ComparatorProvider();
Comparator<Person> comparator = provider::compareByName;
```

Funciones

foreach

```
accountList.forEach(a->a.close());
```

filter

```
Iterable<Account> redAccounts =
  accountList.filter(a->a.balance() < 0 ? true: false);
```

map

```
Iterable<Integer> redBalances =
  accountList.filter(a->a.balance());
```

reduce

```
double total =
  accountList.reduce(0.0,(subtotal,price)-> subtotal+price);
```

Ejercicio

Implementar la clase SquareFunction usando expresiones lambda con funciones

```
public class Calculator {
  public double sum(Serie serie, Function function) {
    double result = 0;
    for (Integer i : serie)
      result += function.apply(i);
    return result;
  }
}
```

Ejercicio resuelto

```
public class Calculator {
  public double sum(Serie serie, Function function) {
    double total =
      serie.reduce(0.0,(subtotal,number)->
        subtotal + function.apply(number);
  }

}
```