Lenguajes de Programación
Tema 4. Paradigma Orientado a Objetos

Java 1.5

Pedro García López

pgarcia@etse.urv.es/

Ejemplo Calculadora

- Se trata de implementar una calculadora que permita calcular operaciones sobre tipos que permitan ser sumados, restados, multiplicados o divididos. Por ejemplo haremos que la calculadora ejecute la operación sumatorio de una lista de valores que se le pasan por parámetro.
- Los valores podrían ser enteros, floats, vectores de enteros, vectores de floats. Matrices de enteros, matrices de float.
- Lo haremos primero con 1.4

Como definir el contrato

- El tipo Operable debe soportar suma, resta, multiplicación y división
- En C++ o Python utilizaríamos la sobrecarga de operadores (+,-,*,/)
- En Java serán métodos normales
- Tomaremos como patrón el método equals:
  - if (a.equals(b)) …
  - public boolean equals(Object obj);

Contrato Operable

```
public interface Operable {
        public Operable suma(Operable otro);
        public Operable resta(Operable otro);
        public Operable multiplicacion(Operable otro);
        public Operable division(Operable otro);
}
```

Código genérico: calculadora

```
public static Operable calcula(List valores, Operable
inicial){
                Iterator it = valores.iterator();
                Operable result = inicial;
                while (it.hasNext()){
                        Operable elem = (Operable)it.next();
                        result = result.suma(elem);
                }
                return result;


        }
```

```
public class PInteger implements Operable {
        private int valor;
        public PInteger(int valor) {
                this.valor = valor;
        }
        public int getValor() {
                return valor;
        }
        public Operable suma(Operable otro) {
                PInteger potro = (PInteger)otro;
                return new PInteger(valor+potro.getValor());
        }
        public String toString(){
                return "("+valor+")";
        }
```

```
public class PDouble implements Operable {
        private double valor;
        public PDouble(double valor) {
                this.valor = valor;
        }
        public double getValor() {
                return valor;
        }
        public Operable suma(Operable otro) {
                PDouble potro = (PDouble)otro;
                return new PDouble(valor+potro.getValor());
        }
        public String toString(){
                return "("+valor+")";
        }
```

```
List penteros = new LinkedList();
penteros.add(new PInteger(1));
penteros.add(new PInteger(7));
penteros.add(new PInteger(8));

List pdoubles = new LinkedList();
pdoubles.add(new PDouble(1.5));
pdoubles.add(new PDouble(7.3));
pdoubles.add(new PDouble(8.8));

PInteger pinicial = new PInteger(0);
PDouble pinicial2 = new PDouble(0.0);

System.out.println(Calculadora.calcula(penteros,pinicial));
System.out.println(Calculadora.calcula(pdoubles,pinicial2))
;
```

```
public class PVector implements Operable {
	private List valores;
	public PVector(List valores) {
			this.valores = valores;
	    }
	public Operable suma(Operable otro) {
	    List result = new LinkedList();
	    PVector potro = (PVector)otro;
	    List listotro = potro.getValores();
	    for (int i=0;i<valores.size();i++){
			Operable elem = (Operable)valores.get(i);
			result.add(elem.suma((Operable)listotro.get(i)));
	    }
	    return new PVector(result);
}
```

Solución con Genericidad

- Ojo: genericidad es polimorfismo paramétrico, código genérico se puede conseguir con el polimorfismo de herencia.

```
public interface Operable<T> {

        public T suma(T otro);
        public T resta(T otro);
        public T multiplicacion(T otro);
        public T division(T otro);

}
```

```
public static <T extends Operable<T>>  T calcula(List<T>
valores, T inicial) {

        T result = inicial;
        for (T elem: valores) {
                result = result.suma(elem);
        }

        return result;


}
```

```
public class PInteger implements Operable<PInteger> {
        private Integer valor;
        public PInteger(Integer valor) {
                this.valor = valor;
        }
        public Integer getValor() {
                return valor;
        }
        public PInteger suma(PInteger otro) {
                return new PInteger(valor+otro.getValor());
        }
```

```
public class PDouble implements Operable<PDouble> {
        private Double valor;
        public PDouble(Double valor) {
                this.valor = valor;
        }
        public Double getValor() {
                return valor;
        }

        public PDouble suma(PDouble otro) {
                return new PDouble(valor+otro.getValor());
        }
```

```
List<PInteger> penteros = new LinkedList<PInteger>();
penteros.add(new PInteger(1));
penteros.add(new PInteger(7));
penteros.add(new PInteger(8));

List<PDouble> pdoubles = new LinkedList<PDouble>();
pdoubles.add(new PDouble(1.5));
pdoubles.add(new PDouble(7.3));
pdoubles.add(new PDouble(8.8));

PInteger pinicial = new PInteger(0);
PDouble pinicial2 = new PDouble(0.0);

System.out.println(Calculadora.calcula(penteros,pinicial));
System.out.println(Calculadora.calcula(pdoubles,pinicial2))
;
```

```
public class PVector<T extends Operable<T>> implements
Operable<PVector<T>> {
        private List<T> valores;
        public PVector(List<T> valores) {
                this.valores = valores;
        }
        public PVector<T> suma(PVector<T> otro) {
                List<T> result = new LinkedList<T>();
                List<T> listotro = otro.getValores();
                for (int i=0;i<valores.size();i++){
                        T elem = valores.get(i);
                        result.add(elem.suma(listotro.get(i)));
                }
                return new PVector<T>(result);

        }
```

Matriz ???

- Se puede implementar creando una clase nueva o parametrizando Pvector
- Asimimos que las operaciones de la matriz son Cij = Aij + Bij
- Sugerencias ?

Ejercicios

- Extender HashSet<E> para que soporte las operaciones union e intersección
- La interfaz Queue<E> tiene operaciones para insertar o sacar elementos de la cola usando normalmente un orden FIFO. Extender la clase LinkedList que cumple Queue, para crear una clase ColaParados que tiene métodos para entrar en la cola, y que cuando alguno sale de la cola llama al método sellar papeles del INEM al parado.
- Experimentar con PriorityQueues en el que el orden lo establece el Comparator