AMPLIACIÓ DE BASES DE DADES I ENGINYERIA DEL PROGRAMARI

Activitat 4: *2n parcial recuperació 2018-2019*

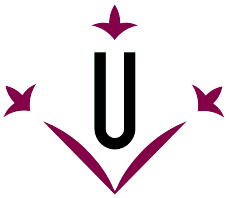
Maria Florencia Martínez Malaret

Jordi Rafael Lazo Florensa

Pere Rollón Baiges

6 de juny de 2021

Grau en Enginyeria Informàtica



Universitat de Lleida
Escola Politècnica Superior

1 Apartat A

Per poder fer que els canvis de valor siguin observables, s'han de fer que la interfície *Expression* siguin una subclasses de*Observable*, per tant, s'ha de convertir en una classe abstracta.

```
import java.util.Observable;

public abstract class Expression <E> extends Observable {
    public abstract E Evaluate();
}
```

En el cas de la classe *Variable*, el que pot provocar el canvi es la crida de *setValue*, per tant, s'afegirà la notificació.

```
public class Variable <E> extends Expression<E>{
    private E value;

    public Variable(E value){
        this.value = value;
    }
    private E evaluate(){
        return value;
    }
    public void setValue(E newValue){
        if(!newValue.equals(value)){
            setChanged();
            notifyObservers(new ValueChanged(value, newValue));
        }
    }
}
```

Pel que fa a la classe *Quantifier* s'afegirà una subexpressió quan es canviï de valor

```
import java.util.ArrayList;
import java.util.List;
import java.util.Observable;
import java.util.Observer;

public abstract class Quantifier<E> extends Expression<E> implements Observer
    {
    private final List<Expression<E>> subExpressions = new ArrayList<>();
    private E oldValue;

    public void addExpression(Expression<E> expression){
        subExpressions.add(expression);
        expression.addObserver(this);
        E newValue = evaluate();
        if(!newValue.equals(oldValue)){
            setChanged();
            notifyObservers(new ValueChanged<E>(oldValue,newValue));
```

```
        }
    }
    public void removeExpression(Expression<E> expression){
        subExpressions.remove(expression);
        expression.deleteObserver(this);
    }

    private E evaluate(){
        E valor = empty();
        for(Expression<E> expression : subExpressions){
            valor = combine(valor, expression.Evaluate());
        }
        return valor;
    }

    public void update(Observable o, Object arg){
        E newValue = evaluate();
        if(!newValue.equals(oldValue)){
            setChanged();
            notifyObservers(new ValueChanged<E>(oldValue,newValue));
        }

    }

    protected abstract E combine(E valor, E evaluate);
    protected abstract E empty();
}
```

Finalment les classes *Concat* i *Max* quedarien de la següent manera:

```
public class Concat extends Quantifier<String> {
    protected String empty(){
        return " ";
    }
    protected String combine(String op1, String op2){
        return op1 + op2;
    }

}
```

```
public class Max extends Quantifier<Integer> {
    protected Integer empty(){
        return Integer.MIN_VALUE;
    }
    protected Integer combine(Integer op1, Integer op2){
        return Math.max(op1,op2);
    }

}
```

El diagrama de classes despés d'implementar el patró *Observer* quedaria:

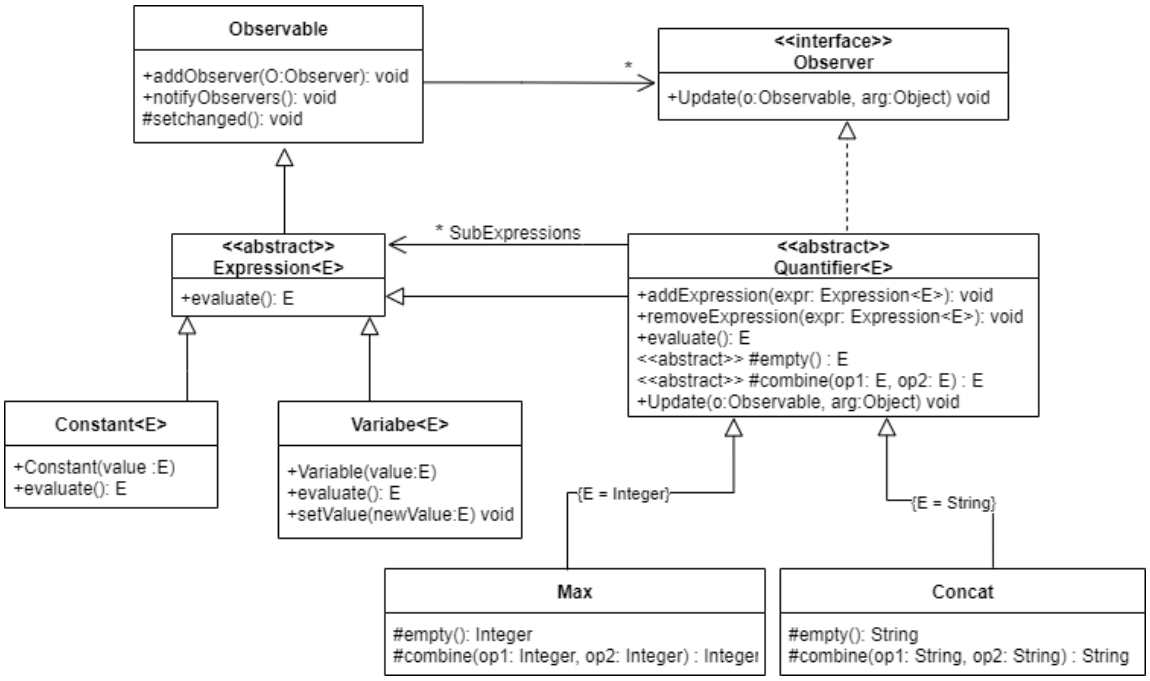


Figure 1: Diagrama de classes resultant de l'implementació *Patró Observable*.

2 Apartat B

Patró estratègia: consisteix en treure els mètodes abstractes *Quantifier* fora de la classe. L'objecte estratègia s'ha de passar al *Quantifier* pel constructor.

Interfície *Estrategia*.

```
public interface Estrategia<E> {
    E empty();
    E combine(E op1, E op2);
}
```

Classe *EstrategiaMax*.

```
public class EstrategiaMax implements Estrategia<Integer> {
    public Integer empty(){
        return Integer.MIN_VALUE;
    }

    public Integer combine(Integer op1, Integer op2) {
       return Math.max(op1,op2);
    }
}
```

Classe abstracta *Expression*.

```
public abstract class Expression<E> {
    public abstract E Evaluate();
}
```

Classe *Max*.

```
public class Max extends Quantifier<Integer> {
    public Max() {
        super(new EstrategiaMax());
    }
}
```

Classe *Quantifier*.

```
import java.util.ArrayList;
import java.util.List;

public class Quantifier<E> implements Expression<E> {
    private final Estrategia<E> estrategia;
    private final List<Expression<E>> subExpressions = new ArrayList<>();

    public Quantifier(Estrategia<E> estrategia) {
        this.estrategia = estrategia;
    }
    public void addExpression(Expression<E> expression){
        subExpressions.add(expression);
    }
    public void removeExpression(Expression<E> expression){
        subExpressions.remove(expression);
    }
    private E evaluate(){
        E valor = estrategia.empty();
        for(Expression<E> expression : subExpressions){
            valor = estrategia.combine(valor, expression.Evaluate());
        }
        return valor;
    }
}
```

El diagrama de classes amb el patró estratègia implementat quedaria de la següent manera:

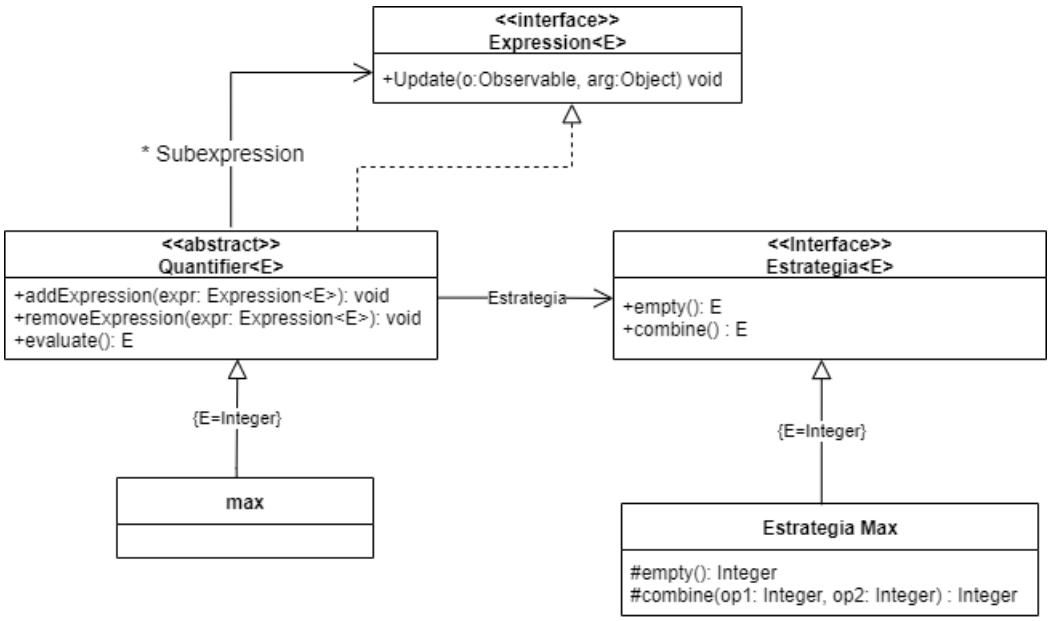


Figure 2: Diagrama de classes resultant de l'implementació *Patró Estratègia*.