calculator\State.java

```java
/**
 * @author  Sebastian Diaz & Guillaume Dunant
 * Date    : 16.11.2023
 * Fichier: State.java
 */
package calculator;

import util.Iterator;
import util.Stack;

/**
 * Classe servant à représenter l'état interne
 * de la calculatrice
 */
public class State {
    private static final String ERROR = "# error #";
    private static final String DEFAULTVAL = "0";
    private static final String NEGATE = "-";
    private static final String DOT = ".";

    private Stack<Double> stack = new Stack<>();
    private Double memory = 0.;

    private String currentVal = DEFAULTVAL;
    private boolean isError = false;
    private boolean userInput = true;

    /**
     * Passe la calculatrice en mode erreur
     */
    public void setError() {
        isError = true;
        currentVal = ERROR;
    }

    /**
     * Enlève le mode erreur de la calculatrice
     */
    public void rstError() {
        isError = false;
        currentVal = DEFAULTVAL;
    }

    /**
     * Obtient l'état du erreur
     * @return true s'il la calculatrice
     * n'est pas en mode erreur
     */
    public boolean noError() {
        return !isError;
    }

    /**
     * Obtient si la valeur courante est une entrée de l'utilisateur
     * ou le résultat d'une opération
     * @return true si c'est ue entrée de l'utilisateur
```

```
57          */
58         public boolean isUserInput(){
59             return userInput;
60         }
61 
62         /**
63          * Change l'état de userInput
64          * @param bool Nouvel état
65          */
66         public void setUserInput(boolean bool){
67             userInput = bool;
68         }
69 
70         /**
71          * Obtient la valeur du dessus de la stack
72          * @return Double
73          */
74         public Double getStackValue(){
75             return stack.pop();
76         }
77 
78         /**
79          * Ajoute un String à la fin de la valeur courante
80          * @param s1 String à ajouter
81          */
82         public void appendToCurrent(String s1) {
83             if (currentVal == null || currentVal.equals(DEFAULTVAL)) {
84                 currentVal = s1;
85             } else if(s1.equals(DOT)) {
86                 if (!currentVal.contains(DOT)) {
87                     currentVal += s1;
88                 }
89             }
90             else {
91                 currentVal += s1;
92             }
93         }
94 
95         /**
96          * Enlève le dernier caractère de la valeur courante
97          */
98         public void removeACharFromCurrent(){
99             currentVal = currentVal.substring(0, currentVal.length() - 1);
100        }
101 
102        /**
103         * Obtient la valeur courante en Double
104         * @return La valeur courante ou null si l'état
105         * est en mode erreur
106         */
107        public Double getCurrent(){
108            if(currentVal.equals(ERROR)){
109                return null;
110            }
111            else{
112                return Double.parseDouble(currentVal);
113            }
114        }
115 
```

```java
    /**
     * Obtient la valeur courante en String
     * @return String
     */
    public String getCurrentInString(){
        return currentVal;
    }

    /**
     * Défini une nouvelle valeur courante
     * @param val Nouvelle valeur
     */
    public void setCurrent(Double val){
        currentVal = val.toString();
    }

    /**
     * Déplace la valeur courante sur la stack
     */
    public void pushCurrent(){
        stack.push(Double.parseDouble(currentVal));
        currentVal = DEFAULTVAL;
    }

    /**
     * Inverse le signe de la valeur courante
     */
    public void negateCurrent(){
        if(currentVal.contains(NEGATE)){
            currentVal = currentVal.substring(1, currentVal.length());
        }
        else{
            currentVal = NEGATE + currentVal;
        }
    }

    /**
     * Stock la valeur courante dans la mémoire
     */
    public void storeInMemory(){
        memory = getCurrent();
        currentVal = DEFAULTVAL;
    }

    /**
     * Modifie la valeur courante par la valeur en mémoire
     */
    public void getMemory(){
        setCurrent(memory);
    }

    /**
     * Retourne les valeurs dans la stack au format String
     * @return String[] contenant les valeurs ou null si la stack est vide
     */
    public String[] getStackInString(){
        if(stack.size() == 0){
            return null;
        }
```

```
175
176         String[] stringStack = new String[stack.size()];
177         Iterator<Double> i = stack.getIterator();
178         int counter = stack.size();
179
180         while(i.hasNext()){
181             stringStack[--counter] = i.next().toString();
182         }
183
184         return stringStack;
185     }
186
187     /**
188      * Réinitialise l'état interne
189      */
190     public void rstState(){
191         rstError();
192         currentVal = DEFAULTVAL;
193         setUserInput(true);
194     }
195
196     /**
197      * Vide la stack
198      */
199     public void emptyStack(){
200         stack.emptyStack();
201     }
202 }
203
```