Contenido

1 Introducción y objetivo del proyecto

Se plantea un proyecto para realizar la calculadora, cuyo enunciado ya ha sido facilitado

Durante el transcurso de su realización observamos una complejidad de desarrollo más que conceptual, lo que aumenta la necesidad temporal para abordarlo, por lo que se proponen alternativas para poder desarrollarlo.

Esta complejidad de la que hablamos crece al aumentar *las posibles entradas* en la calculadora y posibles *modos de operación.*

Exponemos diferentes implementaciones ,

Este documento sólo expresa la máquina de estados y las acciones de la calculadora

Se pueden plantear alternativas a la propuesta inicial, intentando realizar una calculadora de alguno de los tipos propuestos.

Tipo de calculadora		Operaciones	Modo de operar
1.	Calculadora Dora la Exploradora V1	+ - * /	ENTEROS
2.	Calculadora Dora V2	+- * / (+-)	ENTEROS
3.	Calculadora Dora V3	+ - * / (+-) <	ENTEROS
4.	Calcudadora Real sin *retroceso*	+- * / (+-)	REALES
5.	Calculadora Real con *retroceso*	+- * / (+-) <	REALES
6.	Calculadora RealRaciona V1	+- * / (+-)	REAL RACIONAL
7.	Calculadora ReaRacional V2 full	+- * / (+-) <	REAL RACIONAL

2 Entradas

En una primera acción identificamos las posibles entradas y les damos un código.

ENTRADAS A CONSIDERAR			
ENTRADA	**REPRESENTACIÓN**	**VALOR**	**VERSION CALC**
Punto	Pto	.	No en Dora
Separador Racional	SepRa	/	Solo Racional
Numero	Num	[0..9]	Todas
Retroceso	<	←	Solo Retroceso
IGUAL	=	=	Todas
Cambio De signo	+-	±	A partir de Dora V2
Operador Racional	OpRa	+ - * :	Todas
Operador Real	OpRe	+ - * /	Sólo Racional
Clear	C	C	Todas

Realizamos un diagrama y obtenemos los siguientes estados

ESTADO	VERSIÓN DE CALCULADORA
INICIAL	TODAS
OP1	TODAS
OP1_DECIMAL	REAL
DEN_OP1	RACIONAL
ESPERANDO_OP2	TODAS
ESPERANDO_DEN_OP2	RACIONAL
OP2	TODAS
OP2_DECIMAL	REAL
DEN_OP2	RACIONAL
RESULTADO	TODOAS

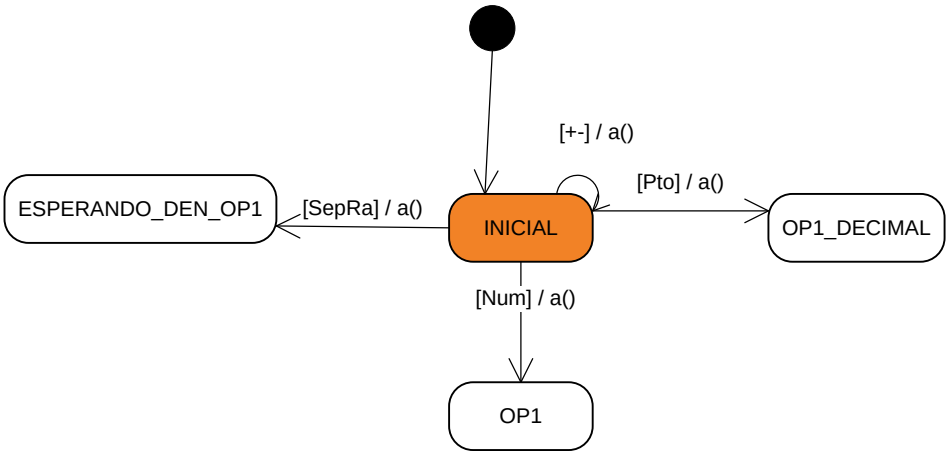
A continuación realizamos un análisis de cada estado

3 Análisis de estados

En función de la calculadora seleccionada, habrá que tener en cuenta los estados correspondientes y no los demás.

Cada transición de estados va etiquetada con la entrada que genera dicha transición y la acción correspondiente.

3.1 INICIAL



A continuación (y esto lo haremos para cada estado), vamos a considerar las entradas que vamos a aceptar en este estado y vemos el estado siguiente, y la acción que desencadenamos.

Las entradas que están en color o sombra, son entradas que no se van a considerar para este estado.

Entrada	Estado Siguiente	Accion
Pto	OP1_DECIMAL	a("**0.**")
SepRa	ESPERANDO_DEN_OP1	a("**0/**")
Num	OP1	a(num)
<	INICIAL	a("")
=		
+-	INCIAL	a("-") | a("")
OpRa		
OpRe		
C	INICIAL	a("")

Si la entrada es +- tenemos dos formas de actualizar, si había un menos lo quitamos, y si no había lo ponemos, como estamos en estado inicial no hay nada más.

Esto que en realidad debería caer en otra clase, por comodidad y simplicidad, se decide hacer en este estado

```
if (valorActual.indexOf('-')==-1)
    accion.actualiza("-");
else
    accion.limpiaPantalla();
```

Con la entrada retroceso como lo único que podíamos haber escrito es un menos, limpiamos la pantalla

En todos los casos hemos identificado una única acción que es actualizar la pantalla con lo que le pasamos como parámetros.

El código

A continuación se lista el código asociado a este estado:

```
private int inicial(Character entrada,String
valorActual,AccionCalculadora accion){
      int estado=this.estado;

      switch (tipoEntrada(entrada)){
        case NUMERO :
            estado=OP1;
            accion.actualiza(valorActual+entrada);
            break;
```

```
        case SEPARADOR_PUNTO :
            estado=OP1_DECIMAL;
            accion.actualiza(valorActual+"0"+entrada);
            break;
        case SEPARADOR_RACIONAL :
            estado=ESPERANDO_DEN_OP1;
            accion.actualiza(valorActual+"0"+entrada);
            break;

        case MAS_MENOS :
            if (valorActual.indexOf('-')==-1)
                  accion.actualiza("-");
            else
                  accion.limpiaPantalla();
            break;
        case RETROCESO:
            accion.actualiza("");
    }
    return estado;

    }
```

Acciones

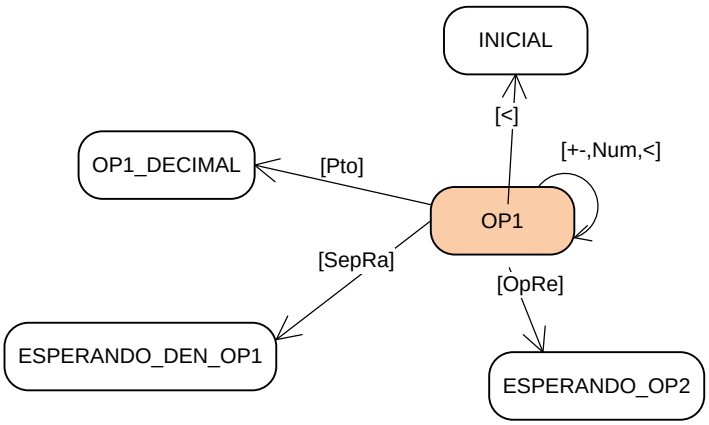
a("") ➔ Representa la acción *actualiza(String).* esta acción lo único que hace es poner en el atributo **resultado** de la clase acción el valor que le pasamos como parámetro. Posteriormente este parámetro es el que la calculadora va a visualizar en la pantalla.

```
/**
* Pone en resultado una cadena de caracteres que le pasamos como parámetro
*  Posteriormente la interfaz retomará este valor para visualizarlo por pantalla
 * @param e
 */
    public void actualiza (String  e){
                this.resultado = e;
                //MRM Pendiente Quitar ceros innecesarios en cada operando
            }
```

3.2 OP1

Este estado genera el siguiente diagrama de estados.

Por comodidad no se indican las acciones, pero se pueden ver en la tabla asociada al diagrama



Entrada	Estado Siguiente	Accion
Pto	OP1_DECIMAL	a(+“**.**”)
SepRa	ESPERANDO_DEN_OP1	a(+“/”)
Num	OP1	a(+num)
<	OP1	quitaCaracterOp1()
	INICIAL	quitaCaracterOp1()
=		
+-	OP1	cambiaSignoOp1()
OpRa	ESPERANDO_ OP2	a(+”/”)
OpRe	ESPERANDO_OP2	a(+”.”)
C	INICAL	a(“”)

Observamos que la única entrada no reconocible en este estado es el =. En cualquier otro caso reaccionamos

Peculiar es la entrada *retroceso* <, esto va a ocurrir en todos los estados, pues esta entrada nos puede llevar a cualquiera de los estados de los que venimos.

Para resolver esta situación se ha optado por interrogar al estado del valor de la pantalla, de este modo podemos analizar de qué estado venimos, posteriormente quitamos el últimos carácter. Habrá algún caso especial que analizaremos en el caso que se produzca.

En este caso podemos volver a uno de los dos estados de los que podemos venir OP1 y INICIAL. El criterio para discernir este hecho lo tomamos de la siguiente manera.

```
int n= Integer.parseInt(valorActual);
if ((n<-9) ||(n>9))
    estado=OP1;
else
    estado =INICIAL;
```

Si el valor es un solo dígito negativo o positivo es que venimos del estado INICIAL otro caso venimos del estado OP1

La acción la misma me sirve para ambos casos, y así mantengo el menos en caso de ser el número negativo. Sólo una observación y es el retorno de este estado me puede llevar al estado INICIAL con el valor 0, pero no crea problemas.

También cuando añadimos un número quitamos posibles ceros de la izquierda.

Acciones

Además de la acción anterior, hemos incorporado 2 acciones más

quitaCaracter() Este método lo que hace es quitar el último carácter del valor actual de la entrada

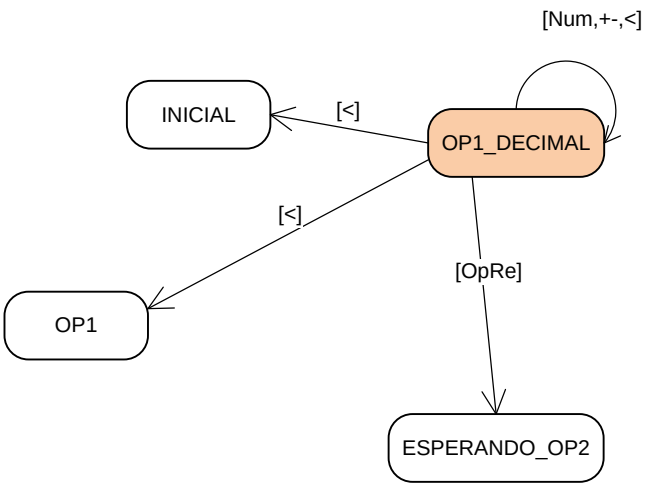
cambiaSignoOp1() Este método está especializado en cambiar el signo del primer operando, indiferentemente de que estemos en modo real o modo racional. En el caso de modo racional el signo afecta a racional entero y no al numerador y denominador independientemente, he creído más interesante hacerlo así pues es el operador al que quiero cambiar el signo.

El código de estos métodos

```
public void quitarCaracterOp1(String s){
            int pos = s.length();
            resultado = s.substring(0,pos-1);

            }
public void cambioSignoOp1(String e){
            if (e.charAt(0)=='-')
                  resultado = e.substring(1);
            else
                  resultado = "-"+e;

            }
```

3.3 OP1_DECIMAL

Diagrama de estados que genera



Entrada	Estado Siguiente	Acción
Pto		
SepRa		
Num	OP1_DECIMAL	a(+num)
<	OP1_DECIMAL	quitaCaracter()
	OP1	quitaCaracter()
	INICIAL	limpiaPantalla()
=		
+-	OP1_DECIMAL	cambiaSignoOp1()
OpRa		
OpRe	ESPERANDO_OP2	a(+OpRe)
C	INICIAL	

En este estado hay varias entradas que no tratamos, como se puede observar en la entrada.

De nuevo se nos presenta la entrada *retorno*, como una entrada que puede llevarnos a tres estados diferentes en función del estado del que venimos.

```
int pos = valorActual.length();
if (valorActual.charAt(pos-1)=='.'){
   if( Integer.parseInt(valorActual.substring(0,pos-1))==0)
       estado = INICIAL;
   else
       estado = OP1;
}
else
       estado = OP1_DECIMAL;
acción.quitarCaracterOp1(valorActual)
```

Jugamos con los caracteres y posiciones para valorar a cuál de los tres estados retornamos.

Vemos que en todos los casos actuamos igual que sería *quitando el último carácter*

De nuevo vemos que al igual que en el estado OP1 si vengo de estado inicial y presiono un pto paso a este estado y veo en la pantalla 0. al quitar el punto se queda el 0

Temporalidad	ESTADO	PANTALLA	E TRADA
0	INICIAL		Pto

1	OP1_DECIMAL	0.	<
2	INICIAL	0	

Pero en pricipio no le damos importancia

Otro tema es no permitir la entrada 9.000000000000+… Se ve en el código como hemos actuado

```
private int op1Decimal (Character entrada, String
valorActual,AccionCalculadora accion){

 int estado=this.estado;
 switch (tipoEntrada(entrada)){

      case NUMERO :
           accion.actualiza(valorActual+entrada);
           break;
      case OPERADOR_REAL :
           estado = ESPERANDO_OP2;
          if (valorActual.charAt(valorActual.length()-1)=='.')
              accion.actualiza(valorActual+"0"+entrada); //9.+ ➔ 9.0+
          else{
              float num = Float.parseFloat(valorActual);//Para evitar
el caso 9.000000000000000+...
              accion.actualiza(""+num+entrada);
           }
           break;
        case MAS_MENOS :
           double n= Double.parseDouble(valorActual);
           accion.actualiza(Double.toString(-n));
           break;
        case RETROCESO :
           int pos = valorActual.length();
           if (valorActual.charAt(pos-1)=='.'){
              if( Integer.parseInt(valorActual.substring(0,pos-1))==0)
                 estado = INICIAL;
              else
                 estado = OP1;
           }
           else
                 estado = OP1_DECIMAL;
           accion.quitarCaracterOp1(valorActual);
           break;
      }
      return estado;

}
```

Acciones

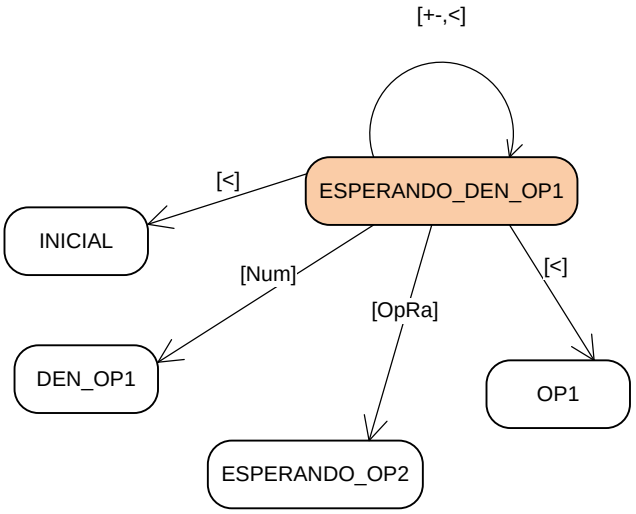
En este estado sólo hemos añadido la acciona **limpiaPantalla()**acciones nuevas a las anteriormente expuestas.

Su código no tiene ningún misterio.

```
public void limpiaPantalla(){
           resultado="";

           }
```

3.4 ESPERANDO_DEN_OP1

El diagrama de estados generado es



Entrada	Estado Siguiente	Acción
Pto		
SepRa		
Num	DEN_OP1	*a(+num)*
<	INICIAL	a(“”) | a(“-“)
	OP1	quitarCaracter()
=		
+-	ESPERANDO_DEN_OP1	cambiarSignoOp1()
OpRa	OP2	a(+’/1’+OpRa)
OpRe		
C		

Es este caso de nuevo la entrada de *retroceso* se lleva la mayor complejidad de todas, pues el resto de entradas resultan sencillas.

El código se expone a continuación básicamente vemos si el número es o no negativo

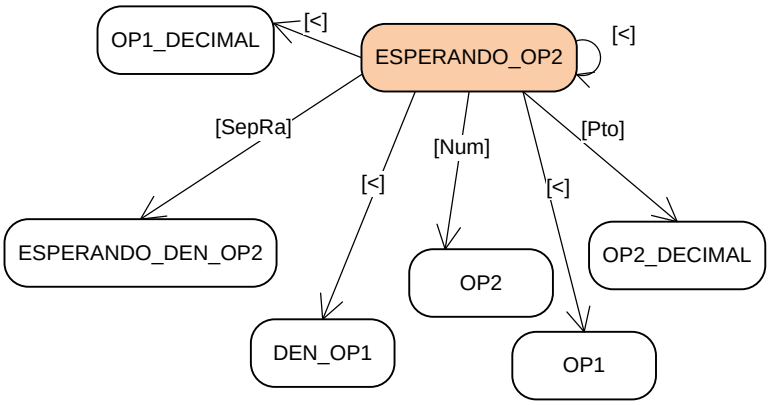
```
      if (valorActual.charAt(0)=='-'){
         if (valorActual.charAt(pos-2)=='0'&pos==3){ //caso -0/
            estado = INICIAL;
            accion.actualiza("-");
         }
         else{
            estado = OP1;
            accion.quitarCaracter(valorActual);   }
      }
      else{
          if (valorActual.charAt(pos-2)=='0'&pos==2){ //caso 0/
              estado = INICIAL;
              accion.actualiza("");    }
          else{
              estado = OP1;
              accion.quitarCaracter(valorActual);
          }
```

A continuación el código generado por este estado

```
private int esperandoDenOp1(Character entrada, String valorActual,
AccionCalculadora accion){
         int estado = this.estado;
            switch (tipoEntrada(entrada)){
            case NUMERO :
                  accion.actualiza(valorActual+entrada);
                  estado=DEN_OP1;
                  break;
            case MAS_MENOS :
                  accion.cambioSignoOp1(valorActual);
                  break;
            case RETROCESO ://PLEASE SIMPLIFICA ESTO MRM
                  int pos = valorActual.length();
                  if (valorActual.charAt(0)=='-'){
                        if (valorActual.charAt(pos-2)=='0'&pos==3){
//caso "-0/" ==> "-"
                           estado = INICIAL;
                         accion.actualiza("-");
                        }
                        else{
                              estado = OP1;
                              accion.quitarCaracterOp1(valorActual);
                        }
                  }
                  else{
                     if (valorActual.charAt(pos-2)=='0'&pos==2){
//caso "0/" ==> ""
                           estado = INICIAL;
                         accion.actualiza("");
                        }
                        else{
                              estado = OP1;
                              accion.quitarCaracterOp1(valorActual);
                        }
                  }
            }
      return estado;

      }
```

3.5 ESPERANDO_OP2

Este estado permite cambiar a muchos estados diferentes según podemos observar en el diagrama, sobre todo por el hecho que podemos llegar a él por muhcos muchos estados diferentes. si no estamos implementando el retroceso esto cambiará bastante.



Entrada	Estado Siguiente	Accion
Pto	OP2_DECIMAL	aCPP(+Pto)
SepRa	ESPERANDO_DEN_OP2	aCPP(+SepRa)
Num	OP2	aCPP(+Num)
<	ESPERANDO_OP2	quitarSignoOp2()
	DEN_OP1	quitarCaracter()
	OP1_DECIMAL	quitarCaracter()
	OP1	quitarCaracter()
=		
+-	ESPERANDO_OP2	cambioSignoOp2
OpRa		
OpRe		
C	INICIO	

En este caso la tecla retroceso nos puede llevar a cuatro estados diferentes, de nuevo analizamos la cadena de caracteres para movernos de estado u realizar acciones

```
int pos = valorActual.length();
String op1;
if (valorActual.charAt(pos-1)==')')
    accion.quitarSignoOp2(valorActual);
else{
    op1=valorActual.substring(0,pos-1);
    accion.quitarCaracter(valorActual);
    if (op1.indexOf('.')!=-1)
        estado = OP1_DECIMAL;
    else
       if (op1.indexOf('/')!=-1)
           estado = DEN_OP1;
       else

          estado =OP1;

}
```

Acciones

Vemos como a la hora de usar el método actualiza, se añaden caracteres además de la cadena y la entrada. Además nuevos métodos que son quitarSignoOp2 y cambiarSignoOp2. Esto es porque se ha decidido que el segundo operando cuando lleva signo se le pone entre paréntesis 9+8 9+(-8) son dos formas posibles de la calculadora, por lo que no va a ser lo mismo modificar el signo del segundo operando, que hacerlo con el primero. Adjuntamos el código de estos nuevos métodos

actualizaConPosibleParentesis(String, char) En este método actualizamos añadiendo el carácter al string teniendo en cuenta que puede haber paréntesis.

```
public void quitarSignoOp2(String s){
            // x + -(ss)   ==> x+ss
      int inicio = s.indexOf('(');
      int fin = s.length();
      resultado = s.substring(0,inicio-2)+s.substring(inicio+1,fin);

}
public void cambiarSignoOp2(String e,int tipo){
      int operador ;
      operador = getPosicionOperador (e,tipo);
      if (e.contains("("))
         resultado= e.substring(0,e.indexOf('('))
                    +e.substring(e.indexOf('(')
                    +2,e.indexOf(')'));
      else
        resultado=e.substring(0,operador+1)
                  + "(-"
                  +e.substring(operador+1)
                  + ")";

      }
      public void actualizarConPosibleParentesis(String p, String e){
            int pos = p.length();
            if (p.contains(")")){
                  resultado=(p.substring(0,pos-1)+e+")");
                System.out.println("Resultado "+resultado);
            }
            else
                  resultado = p+e;

            }
```

El código de este estado queda como se expresa a continuación

```
private int esperandoOp2(Character entrada, String
valorActual,AccionCalculadora accion){
      int estado=this.estado;
      switch (tipoEntrada(entrada)){
        case NUMERO :
            estado = OP2;
            accion.actualizarConPosibleParentesis (valorActual,
                                          Character.toString(entrada));
            break;
         //Ante la entrada punto o / no he de ver si estoy en tipo
REAL o RACIONAL ya que sÃ³lo existe
            //la entrada punto en tipo REAL :)
        case SEPARADOR_PUNTO :
            estado=OP2_DECIMAL;
            accion.actualizarConPosibleParentesis( valorActual,"0"
                                                  +entrada);
```

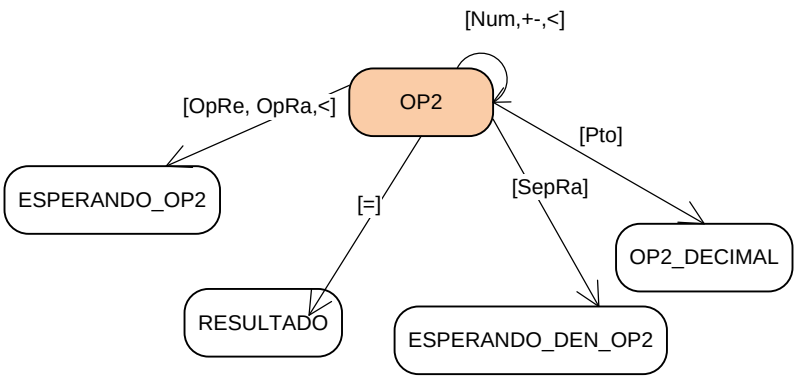
```
            break;
         case SEPARADOR_RACIONAL :
            estado=ESPERANDO_DEN_OP2;
            accion.actualizarConPosibleParentesis(valorActual,"0"
                                                    +entrada);
            break;

        case MAS_MENOS :
            if (valorActual.contains("("))
                  accion.actualiza(valorActual.substring(0,
                                        valorActual.indexOf('(')));
            else
                  accion.actualiza(valorActual+"(-)");
            break;

        case RETROCESO :
            int pos = valorActual.length();
            String op1;
            if (valorActual.charAt(pos-1)==')'){
                  accion.quitarSignoOp2(valorActual);
                  System.out.println("Esperando Op2 retroceso"
                                    +valorActual);
            }
            else{
                  op1=valorActual.substring(0,pos-1);
                  accion.quitarCaracterOp1(valorActual);
                if (op1.indexOf('.')!=-1)
                      estado = OP1_DECIMAL;
                else
                  if (op1.indexOf('/')!=-1)
                        estado = DEN_OP1;
                    else
                        estado =OP1;
            }
            break;
      }
      return estado;
}
```

3.6 OP2

En este estado ya tenemos el primer operando y el operador

Ya hemos comentado que el signo menos en este operador va a ir entre paréntesis (- op).

En este estado ya podemos realizar cáculos y por lo tanto realizar la operación.



Entrada	Estado Siguiente	Accion
Pto	OP2_DECIMAL	aCPPP(valorActual,Pto)
SepRa	ESPERANDO_DEN_OP2	aCPPP(valorActual,SepRa)
Num	OP2	aCPPP(valorActual,Num)
<	OP2	quitarCaracterOp2()
	ESPERANDO_OP2	quitarCaracterOp2()
=	RESULTADO	calcula()
+-	OP2	cambiarSignoOp2()
OpRa	ESPERANDO_OP2	Calcula() addResultado(OpRa)
OpRe	ESPERANDO_OP2	Calcula() addResultado(OpRa)
C	INICIAL	a(“”)

El retroceso se soluciona de la siguiente manera

```
int c = accion.caracteresOp2(valorActual,tipo);
if (c==0)
   estado = ESPERANDO_OP2;
else
   estado = OP2;

accion.quitarCaracterOp2(valorActual);
```

Para gestionar el retroceso volvemos a utilizar el método llamado **caracteresOp2(String, tipo)**

aCPP()➔ actualizaConPosibleParentesis() ➔ Este método tiene en cuenta que es posible que hubiera paréntesis en el segundo parámetro, en cuyo caso he de añadir el nuevo carácter dentro del paréntesis

En Este caso observamos que todas las entradas son posibles, y la calculadora reaccionará en consecuencia

En el caso de retroceso en este caso tenemos dos posibilidades que implementamos mirando el número de caracteres que tiene el segundo operando, teniendo dos posibilidades, que tengo un solo carácter o que tenga dos. Para ello creamos una función que nos devuelva el número de caracteres del segundo operando

```
        int c = caracteresOp2(valorActual);
        if (c==0)
           estado = ESPERANDO_OP2;
        else
           estado = OP2;

        accion.quitarCaracterOp2(valorActual);
```

Accion

En este caso hemos añadido dos métodos para realizar y visualizar el resultado de la operación

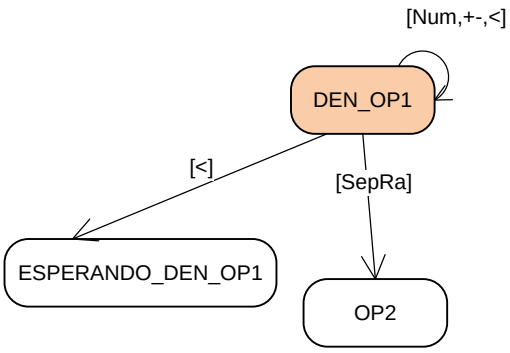
addResultado(char) la idea es añadir el operador al resultado

```
public void addResultado(char e){
                    resultado +=e;

              }
```

calcula(Strint, tipo) este método realiza el cáculo a partir de un string que contiene una operacion

```
public void calcula(String valorActual, int tipo){
  if (tipo==REAL){
       System.out.println("1En calcula con operacion "+valorActual);
       operacion.setOperacion(valorActual);
       resultado=operacion.resultado().toString();

  }else{
       System.out.println("2En calcula con operacion "+valorActual);
       operacionR.setOperacion(valorActual);
       resultado=operacionR.resultado().toString();
       System.out.println("Valor de "+valorActual +
                          "Resultado =  "+resultado);
       }

}
```

3.7 DEN_OP1



Entrada	Estado Siguiente	Accion
Pto		
SepRa		
Num	DEN_OP1	a(+Num)
<	ESPERANDO_DEN_OP1	quitaCaracter()
	DEN_OP1	quitaCaracter()
=		
+-	DEN_OP1	camibaSignoOp1()
OpRa	ESPERANDO_OP2	a(+OpRa)
OpRe		
C	INICIAL	a(“”)

En esta caso vemos más simplificado el esquema.

El tema de retroceso lo resolvemos con el siguiente algoritmo

```
int pos = valorActual.length();
if (valorActual.charAt(pos-2)=='/')
   estado = ESPERANDO_DEN_OP1;
accion.actualiza(valorActual.substring(0,pos-1));
break;
```

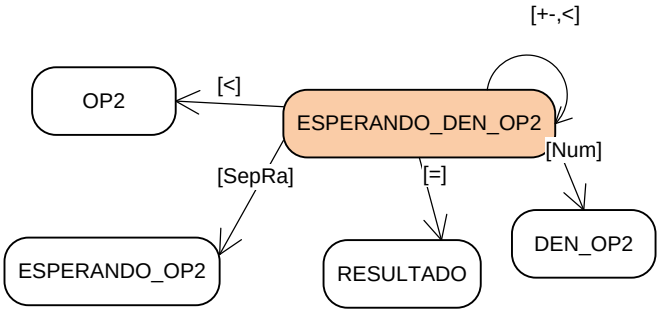
Ahora no hemos añadido ningún método nuevo para las acciones

El código de este estado:

```
private int denOp1(Character entrada, String valorActual,
AccionCalculadora accion){
        int estado = this.estado;
        switch (tipoEntrada(entrada)){

           case NUMERO :
                 accion.actualiza(valorActual+entrada);
                 break;
           case OPERADOR_RACIONAL :
                 estado = ESPERANDO_OP2;
                 accion.actualiza(valorActual+entrada);
                 break;

           case MAS_MENOS :
                 accion.cambioSignoOp1(valorActual);
                 break;
```

```
            case RETROCESO :
                  int pos = valorActual.length();
                  if (valorActual.charAt(pos-2)=='/')
                              estado = ESPERANDO_DEN_OP1;
                  accion.actualiza(valorActual.substring(0,pos-1));
                  break;
      }
return estado;

}
```

3.8 ESPERANDO_DEN_OP2



Entrada	Estado Siguiente	Accion
Pto		
SepRa		
Num	DEN_OP2	*a(+num)*
<	OP2	quitaCaracterOp2()
	ESPERANDO_OP2	quitaCaracterOp2()
=	RESULTADO	Calcular()
+-	ESPERANDO_DEN_OP1	cambiaSignoOp2()
OpRa	ESPERANDO_OP2	Calcula() addResultado(+OpRa)
OpRe		
C		

El caso de retroceso es exactamente igual que en caso de OP2, y lo resolvemos

de igual manera

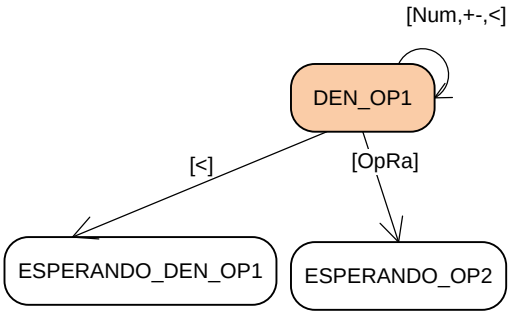
```
int c = caracteresOp2(valorActual);
      if (c==1)
         estado = ESPERANDO_OP2;
      else
         estado = OP2;

      accion.quitarCaracterOp2(valorActual);
```

El código de este estado

```
private int esperandoDenOp2(Character entrada, String valorActual,
AccionCalculadora accion){
         int estado = this.estado;
            switch (tipoEntrada(entrada)){

            case NUMERO :
                  estado = DEN_OP2;
                  accion.actualiza(valorActual+entrada);
                  break;
            case MAS_MENOS :
                  accion.cambiarSignoOp2(valorActual,tipo);
                  break;
            case IGUAL:
                  if (valorActual.contains("("))
                           valorActual=valorActual.substring( 0,
                                          valorActual.length()-2)
                                           +"1)";
                  else
                        valorActual=valorActual+"1";

                  accion.calcula(valorActual, tipo);
                  estado = RESULTADO;
                  break;
            case OPERADOR_RACIONAL:
                  if (valorActual.contains("("))
                           valorActual=valorActual.substring( 0,
                                          valorActual.length()-2)
                                                   +"1)";
                  else
                        valorActual=valorActual+"1";
                  accion.calcula(valorActual, tipo);
                  accion.addResultado(entrada);
                  estado=ESPERANDO_OP2;
                  break;
            case RETROCESO ://MRM Me quede aquÃ-
                  //  1 a/b op c/   ==> a/b op c    OP2
                  //  2 a/b op 0/   ==> a/b op       ESPERANDO_OP2
                  //  3 a/b op (-c/)   ==> a/b op (-c)    OP2
                  //  4 a/b op (-0/)   ==> a/b op       ESPERANDO_OP2

                        int c = accion.caracteresOp2(valorActual,tipo);
                  if (c==1)
                     estado = ESPERANDO_OP2;
                  else
                        estado = OP2;
```

```
                    accion.quitarCaracterOp2(valorActual);
                        break;
                }//End case
        return estado;
}
```

3.9 DEN_OP2



Entrada	Estado Siguiente	Accion
Pto		
SepRa		
Num	DEN_OP2	*a(+num)*
<	DEN_OP2	aRCPP()
	ESPERANDO_DEN_OP2	aRCPP()
=	RESULTADO	Calcular()
+-	DEN_OP1	cambiaSignoOp2()
OpRa	ESPERANDO_OP2	Calcula() addResultado(+OpRa)
OpRe		
C		

En este caso el retroceso queda

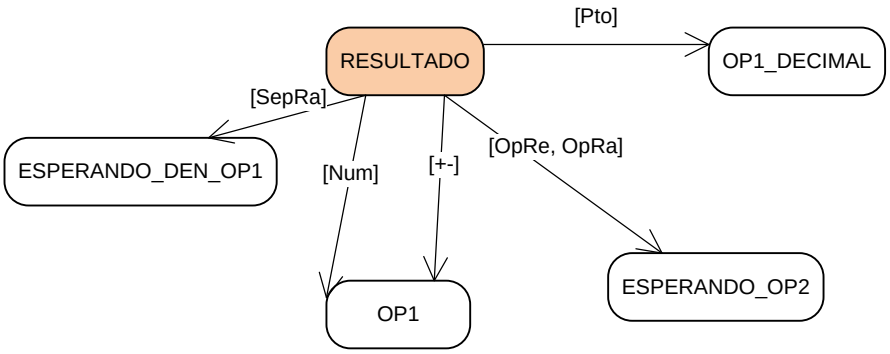
```
// 4 a/b op -(c/de) ==> a/b op -(c/d) DEN_OP2
      int c = accion.caracteresOp2(valorActual,tipo);
      if (c==0)
         estado = ESPERANDO_OP2;
      else
          estado = OP2;

      accion.actulizaRetrocesoConPosibleParentesis(valorActual);
```

El código de este estado:

```
private int denOp1(Character entrada, String valorActual,
AccionCalculadora accion){
         int estado = this.estado;
         switch (tipoEntrada(entrada)){

            case NUMERO :
                  accion.actualiza(valorActual+entrada);
                  break;
            case OPERADOR_RACIONAL :
                  estado = ESPERANDO_OP2;
                  accion.actualiza(valorActual+entrada);
                  break;

            case MAS_MENOS :
                  accion.cambioSignoOp1(valorActual);
                  break;

            case RETROCESO :
                  int pos = valorActual.length();
                  if (valorActual.charAt(pos-2)=='/')
                              estado = ESPERANDO_DEN_OP1;
                  accion.actualiza(valorActual.substring(0,pos-1));
                  break;
         }
      return estado;

      }
```

3.10 RESULTADO

El diagrama de estado



Entrada	Estado Siguiente	Accion
Pto	OP1_DECIMAL	a(Pto)
SepRa	ESPERANDO_DEN_OP1	a(SepRa)
Num	OP1	a(Num)
<		
=		
+-	OP1	cambiarSigno();
OpRa	ESPERANDO_OP2	addResultado(opRa)
OpRe	ESPERANDO_OP2	addResultado(opRe)
C	INICIO	A(“”)

Observamos una situación muy similar a la que tendríamos en el estado inicial. Añadimos a ella la posibilidad de llevarnos el valor del resultado como primer operando a una nueva operación.

En este caso no vamos a considerar el retroceso como una entrada permitida ☺

El código asociado a este estado

```
private int resultado(Character entrada, String valorActual,
AccionCalculadora accion){
        int estado = this.estado;
                switch (tipoEntrada(entrada)){

                case NUMERO :
                      estado=OP1;
                      accion.actualiza(""+entrada);
                      break;
                case OPERADOR_RACIONAL :
                case OPERADOR_REAL :
                      estado=ESPERANDO_OP2;
                      accion.addResultado(entrada);
                      break;
                case MAS_MENOS :
                      estado = OP1;
```

```
                        accion.cambiaSignoResultado();
                        break;

                case CLEAR :
                }
            return estado;

    }
```

4 Acciones

Las acciones se detallan en cada estado que aparecen