Comentarios a la **solución** del ejercicio voluntario propuesto en 3º evaluación (2ª fase), que decía:

El ejercicio consiste en implementar una calculadora:

- El usuario introduce pares de número y operador (pulsa *enter* para introducirlos)
  - Los números son double
  - Los operadores aceptados son los siguientes caracteres:
    - \+ para sumar
    - \- para restar
    - * para multiplicar
    - / para dividir
    - = para indicar terminar de introducir datos. El programa debe entonces calcular el resultado según el ejemplo:
      `"36 + 4 - 10 * 2 / 10 =" dará 6`

Así que

1. Hay que asegurarse de que el usuario introduce números cuando tiene que introducir números y los caracteres correspondientes a los operadores indicados arriba (sólo esos!).
2. Encolar los pares (número, operador), que llamaremos datos (a partir de ahora)
3. Detectar el operador '=' que provocará que dejemos de encolar datos y calculemos el resultado de las operaciones. Para ello, debemos
   a) Contar con una variable double que almacene el resultado (de momento transitorio).
   b) Desencolar los datos almacenados uno a uno y operarlos con el resultado (anterior) respetando el orden: **resultado <operación> nueva_cantidad** siendo <operación> la indicada en el dato previamente desencolado, y nueva_cantidad el valor numérico del dato recién desencolado.
   c) Al desencolar el primer dato (caso especial) simplemente se guarda su valor numérico en resultado.
   d) Esta operación actualizará el valor de la variable resultado para el siguiente proceso desencolado + operación
   e) Cuando ya no queden más datos que desencolar (se puede comprobar que '=' es el operador del último dato desencolado), se dará el resultado como definitivo por pantalla.
   En el ejemplo dado, la secuencia (desencolado y operaciones) sería:

• 36	+	resultado = 36		operación que guardo +
• 4	-	resultado = 36 + 4	= 40	operación que guardo -
• 10	*	resultado = 40 - 10	= 30	operación que guardo *
• 2	/	resultado = 30 * 2	= 60	operación que guardo /
• 10	=	resultado = 60 / 10	= 6	fin: saco el resultado = 6

Valoración:

- El control de I/O manejando excepciones para advertir de un eventual despiste al usuario (que introduce una letra en lugar de un número o un carácter que no corresponde a ningún operador válido) sin que el programa se pare y dé los errores del compilador JAVA. Asimismo, se detectará el caso de división por cero y se dará un mensaje apropiado (no se ofrecerá entonces resultado alguno) → **3,5 puntos**
- Diseño de clases e implementación de la cola → **2,5 puntos**
- Implementación de métodos (incluyendo el main) para realizar los cálculos como se ha descrito → **3,5 puntos**
- **0,5 puntos** de valoración global (planteamiento, comentarios en código, facilidad para seguirlo, etc.).

Nota: Se facilita la interfaz Iqueue y las clases SNode, Dato y LeerTeclado. Se recomienda que el método main sólo conste de un bucle (tipo while) que vaya aceptando datos hasta que se incluya el operador '='

1. Siguiendo el esquema propuesto, elaboramos la clase ColaOperaciones con elementos node del tipo Dato:

```
public class ColaOperaciones<Dato> implements IQueue<Dato> {
        public SNode<Dato> front, tail;
        public int size;
```

   E implemento (codifico) los métodos de la interface Iqueue:

```
E dequeue();
void enqueue(E e);
E front();
int size();
boolean isEmpty()
```

2. En la clase principal (contiene el método main), que llamaremos simplemente Calculadora, declaramos la vaiable

```
static ColaOperaciones operacionesPendientes = new ColaOperaciones ();
```

   Y, en el método main, pondré las variables

```
double resultado = 0.0;
Dato operandos = new Dato();
```

   de modo que el programa funcione con un bucle infintio de entrada de operandos hasta que la componente 'operador' del operando introducido sea '=':

```
while (operandos.operador != '='){
        errores = 3;
        operandos = EntradaOperaciones.operacion (errores);
        if (EntradaOperaciones.ok)
                {operacionesPendientes.enqueue(operandos);}
}
resultado = calcularResultado();
```

3. Sólo tengo que codificar el método *calcularResultado*() y chequear que *operación* (de la clase Dato) ha sido formada correctamente. Para esto, utilizo la variable errores como límite admisible de equivocaciones del usuario cuando introduzca un número o un operador.

   a) Por ejemplo, para validar la entrada de operadores (tipo char de los que sólo admitimos los signos aritméticos), podemos hacerlo con un método recursivo (hasta que se supere el límite de intentos dado por la variable errores):

```
static char nuevo_operador (int intentos){
        char operador = '=';
        int cont = intentos;
        try {operador = (char) System.in.read();}
        catch (IOException e) {cont --;}
        if(operador != '+' && operador != '-' && operador != '*' &&
                        operador != '/' && operador != '=') cont --;
        //System.out.println("contador de intentos de operador = " +cont);
        if (cont >0 && cont <intentos) {
                System.out.print("Operador? ");
                return nuevo_operador(cont);
        }
        if (cont <=0) ok = false;
        return operador;
}
```

De modo que, cada fallo de entrada reduce cont pero, si aún no llega a 0, el usuario puedo volver a intentarlo (se le advierte del fallo con el mensaje “Operador?”). Si se supera el límite de errores admisibles, se pone a *false* el centinela ok para que el dato formado no sea encolado.

   b) Igualmente se trabaja con la entrada de valores numéricos de cada dato. En el siguiente ejemplo, se ha utilizado la entrada de datos a String a partir del Scanner del stream System.in

```
static double nuevo_valor (int intentos){   // recibe el máximo número de
                errores que el usuario puede cometer al introducir un dato
        int fallos_permitidos = intentos;
        Scanner entrada = new Scanner (System.in) ;   //Scanner de la
                                                entrada estándar (teclado)
        String cantidad = "";
        double valor = 0.0;
        cantidad = entrada.nextLine();     // tomo los inputs como Strings
        try{             // validamos si el valor introducido es numérico
                valor=Double.parseDouble(cantidad);
        }
        catch(java.lang.NumberFormatException e){   // errores de I/O no
                                                convertibles a double
                fallos_permitidos--;        // Si el usuario NO introduce
                                cifras, se le cuenta 1 error de entrada
        }
```

De nuevo, la variable fallos_permitidos se decrementa cada vez que el usuario introduce algo que no corresponde a un valor numérico (double). Si aún no ha llegado al límite, se utiliza el mismo método (recursividad) para ofrecer al usuario una nueva oportunidad, etc.:

```
        if (fallos_permitidos>0 && fallos_permitidos < intentos) {
                System.out.print("número?");
                return valor = nuevo_valor (fallos_permitidos);    // nueva
                                oportunidad de introducir el valor numérico
        }
```

4. La codificación del método *calcularResultado()* simplemente sigue el algoritmo

   Desencolar los datos almacenados uno a uno y operarlos con el resultado (anterior) respetando el orden: **resultado <operación> nueva_cantidad** siendo <operación> la indicada en el dato previamente desencolado, y nueva_cantidad el valor numérico del dato recién desencolado. Es decir:

```
public static double calcularResultado(){
   double calculo = 0.0;
   char oper_pte = '?';
   Dato valores_desencolados = new Dato();
   while (operacionesPendientes.size()>0){
       valores_desencolados = (Dato) operacionesPendientes.dequeue();
       //System.out.println( "dato desencolado: " +
                                                  valores_desencolados.toString());
       switch (oper_pte){
          case '?' : {calculo = valores_desencolados.valor;break;}
          case '+' : {calculo = calculo + valores_desencolados.valor; break;}
          case '-' : {calculo = calculo - valores_desencolados.valor; break;}
          case '*' : {calculo = calculo * valores_desencolados.valor; break;}
          case '/' : {
              if (valores_desencolados.valor != 0.0) calculo = calculo /
                                                 valores_desencolados.valor;
              else {
                 System.err.println("Intento de división por cero!");
                 EntradaOperaciones.ok = false;
              }
              break;
          }
       }
       oper_pte = valores_desencolados.operador;
             //System.out.println("resultado provisional = "+ calculo +
                                      "\noperación pendiente: " + oper_pte);
       }
       return calculo;
}
```

Nótese que es preciso hacer casting a tipo Dato de los valores que desencolamos ya que, en principio son tipo object.

5. El error "división por cero" se controla en este paso (podría hacerse de otro modo) y no se saca "*infinity*". Este error impide ofrecer un resultado así que ponemos a *false* el centinela ok.

   En el main (tras la sentencia `resultado = calcularResultado()`, pondremos

```
       if (EntradaOperaciones.ok) {
          System.out.println("Resultado: "+ resultado);
          System.out.println("\n\nGracias por utilizar nuestra calculadora.");
       }
       else System.err.println("\nERROR!\tNo podemos ofrecer un resultado");
```

El ejercicio queda así resuelto con 3 clases (Calculadora, ColaOperaciones y EntradaOperaciones) y 3 métodos: nuevo_valor (int intentos), nuevo_operador (int intentos) y calcularResultado(): Se va encolando operandos validados hasta que el usuario pide el resultado (introduce el signo '=') . Entonces se desencolan y, con ellos, calculamos el resultado aritmético.