

Proyecto 1:

ADT Stack

Asignatura:

Profesor:

Autor:

Documento

Pruebas de Software

Iván Zuñiga Valenzuela

Jorge Cisterna Calderón

Versión 1.0

Contenido

Tabla de contenido

Introducción..... 3

 TDA Pila (STACK).....3

 Operadores de Pilas..... 4

Objetivo del Proyecto..... 5

Proyecto 1.....6

 1. Propuesta de Desarrollo con Énfasis en las Pruebas..... 7

 2. Plan de Pruebas..... 7

 3. Desarrollo con enfoque TDD.....9

Introducción

Un tipo de dato abstracto (TDA) es un modelo matemático, junto con varias operaciones definidas sobre ese modelo. Los TDA se representan en función de los tipos de datos y los operadores manejados por ese lenguaje.

Para representar los TDA se usan estructuras de datos que están constituidas por conjuntos de variables (a veces de diferente tipo).

Notemos el siguiente fragmento.

```
int x, y;
{
    x = 3;
    y = x + 2;
}
```

Decimos que int es un tipo del lenguaje y que x e y son de tipo int. Es decir que contienen valores de tipo int. El número 3 es almacenado. La expresión $x + 2$ evalúa un valor entero.

Dado que x e y contienen valores enteros matemáticos, entonces ciertas operaciones matemáticas son permitidas: $+$ y $-$. Para otros tipos como reales, void, boolean aplica lo mismo.

Definición: Un tipo de dato (type) consiste en un conjunto de valores (dominios) y un conjunto de operaciones.

Definición: Un literal es una constante cuyo valor está dado por su forma escrita. Los tipos se pueden clasificar en: Escalar y Estructurado.

Definición: Un escalar es un tipo cuyo dominio consiste en entidades atómicas. Un escalar no contiene partes que puedan ser accedidas o manipuladas en forma independiente: integer, char...

Definición: Un tipo estructurado tiene un dominio consistente de valores estructurados constituidos de componentes escalares (estructurados): array.

El arreglo es la representación computacional de la estructura matemática denominada vector. Podemos pensar el arreglo como una función del índice.

$$f(i) = v$$

en que i es el índice y v el valor contenido en la variable que compone la regla. Esta variable tiene un tipo definido.

```
int valores[28];
tipo nombre variable[tipo];
```

TDA Pila (STACK)

Un stack es un tipo especial de la lista en que todas las inserciones y borrados toman lugar en un sólo lugar llamado tope. Otro nombre que recibe el stack es "push down list" y LIFO: Last In First Out. Un modelo intuitivo es una pila de platos, en que conviene mover el de más arriba.

Operadores de Pilas

S: es una pila de objetos de un tipo dado (`tipoElemento` o `typeElement`).

x: es un objeto.

`MakeNull (S) :` Hace de S un stack vacío.

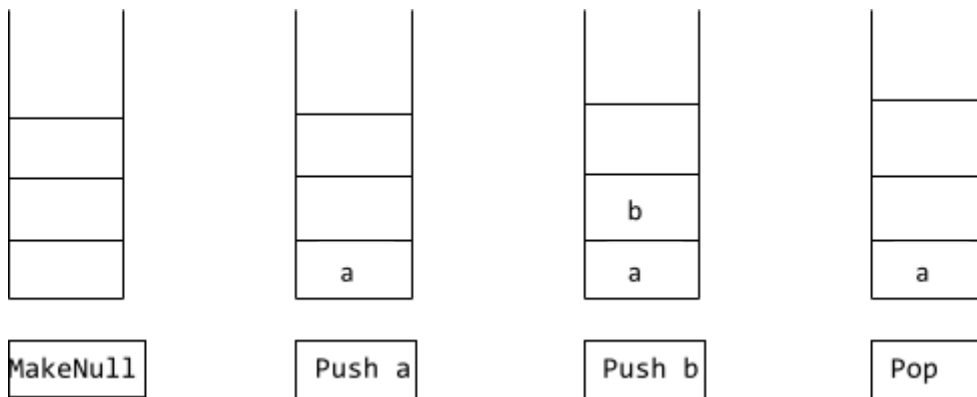
`Top (S) :` Retorna el elemento al tope del stack.

`Pop (S) :` Saca el elemento del tope del Stack.

`Push (x, S) :` Inserta el elemento x en el tope del Stack.

`Empty (s) :` Retorna TRUE si S está vacío; FALSE en otro caso.

En forma gráfica podemos representar el comportamiento de un stack como sigue:

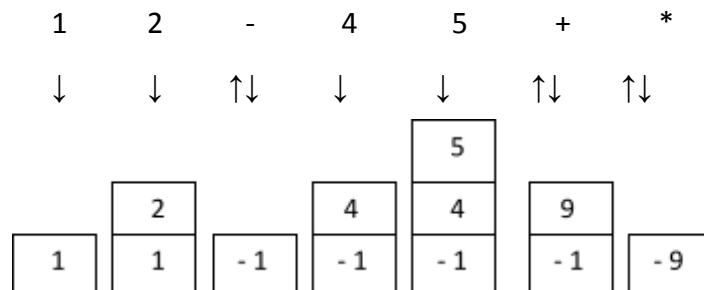


Objetivo del Proyecto

Implemente el TDA Stack, que permita llevar a cabo el funcionamiento de una calculadora RPN. Esta calculadora es del tipo HP. La calculadora clásica es de tipo notación infija en que una expresión $(1-2) * (4+5) =$ se recorre en forma serial y requiere paréntesis.

Esta expresión en notación polaca inversa (RPN) es $1\ 2\ -\ 4\ 5\ +\ *\ =$. El signo $=$ es opcional, los paréntesis no son necesarios.

La idea es que cada operando se almacene en un stack. Cuando llega un operador, se extrae el número de operandos apropiados (2 para operaciones binarias). Luego se aplica el operador y el resultado se devuelve a la pila.



Un posible algoritmo que resuelve el funcionamiento de este tipo de calculadoras es el que se propone a continuación:

```
Mientras (el siguiente operador u operando no sean fin)
  Si (es número)
    Introducirlo al stack
  Sino
    Si (es operador)
      Sacar operandos
      Hacer operación
      Introducir resultado en el stack
    Sino
      Error
```

Proyecto 1

1. Formar grupos de dos personas
2. Aplicar Programación eXtrema para llevar a cabo el objetivo del proyecto
3. Generar y presentar una propuesta de desarrollo, breve, con énfasis en la pruebas
4. Generar y presentar el Plan de Pruebas
5. Generar la solución
6. Generar la evidencia de las pruebas
7. Subir documentación y solución a Moodle

1. Propuesta de Desarrollo con Énfasis en las Pruebas

Para la solución se usarán las siguientes tecnologías:

Lenguaje de programación: Javascript (ES6+)

Frameworks: Jest (para las pruebas unitarias en el enfoque TDD), NodeJS.

El proceso de desarrollo se llevará a cabo en los siguientes pasos:

1. Definir las pruebas necesarias para verificar que la funcionalidad del TDA Stack y la calculadora RPN sean correctas.
2. Escribir pruebas unitarias que validen el comportamiento de las operaciones de la calculadora RPN y las funcionalidades del Stack.
3. Basándose en las pruebas, se implementará el Stack y la lógica de la calculadora RPN, partiendo siempre desde el código mínimo necesario para aprobar los tests.
4. Ejecutar las pruebas para confirmar que las funcionalidades están correctamente implementadas.
5. En caso de que se haya omitido alguna prueba en la definición inicial, se agregará en este paso.

2. Plan de Pruebas

Las siguientes pruebas serán para cubrir las funcionalidades clave de la calculadora RPN:

Pila / Stack:

Estas pruebas serán para desarrollar y comprobar el correcto funcionamiento de la Pila o Stack.

- Prueba de creación de pila vacía (IsEmpty)
 - Prueba de apilado (Push):
 - Prueba de desapilado (Pop):
 - Prueba de consulta del valor superior (checkLast):
 - Prueba de vaciado de pila (clear):
-

Calculadora RPN:

Estas pruebas serán para desarrollar y comprobar el correcto funcionamiento de la Calculadora RPN.

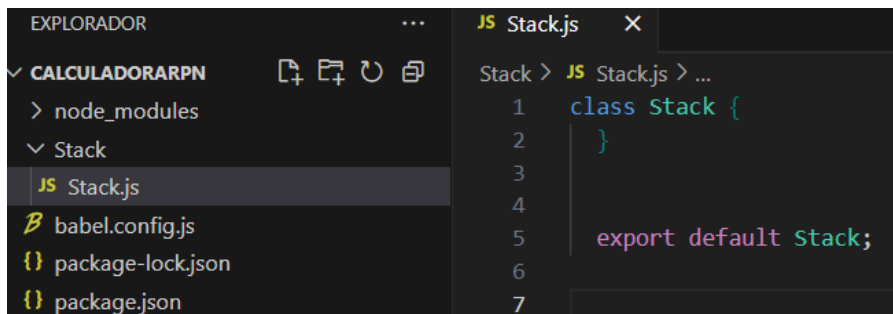
- Suma de enteros
 - Resta de enteros
 - Multiplicación de enteros
 - División de enteros
 - Suma de números decimales
 - Operación inválida (carácter no válido)
 - División por cero
 - Caracteres no numéricos ni operadores
 - Expresión con operandos insuficientes
 - Expresión con demasiados operandos
 - Número grande
 - Número muy pequeño
 - Exponente de enteros
 - Expresión compleja con varias operaciones
-

3. Desarrollo con enfoque TDD

Pruebas:

Para el desarrollo guiado por prueba estaremos usando **Jest** que es un framework de código abierto usado para realizar pruebas.

Creemos la primera clase, que será la clase de la Pila o Stack. Esta clase estará vacía para asegurarnos que al ejecutar la primera prueba no de un falso positivo.

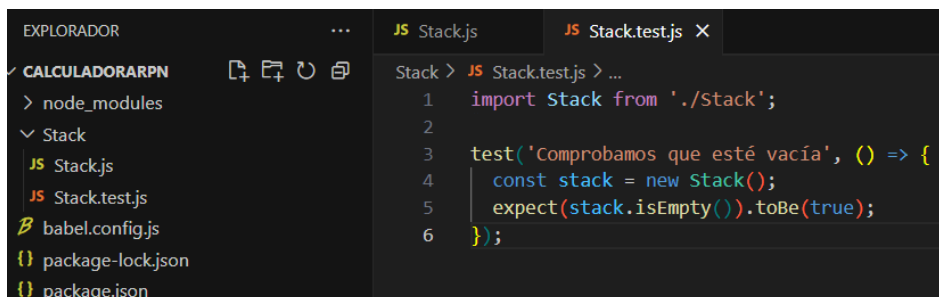


The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows a project named 'CALCULADORARP' with a folder 'Stack' containing 'Stack.js'. The main editor shows the content of 'Stack.js' with the following code:

```
Stack > JS Stack.js > ...
1 class Stack {
2   }
3
4
5   export default Stack;
6
7
```

y creamos el siguiente Test para la clase:

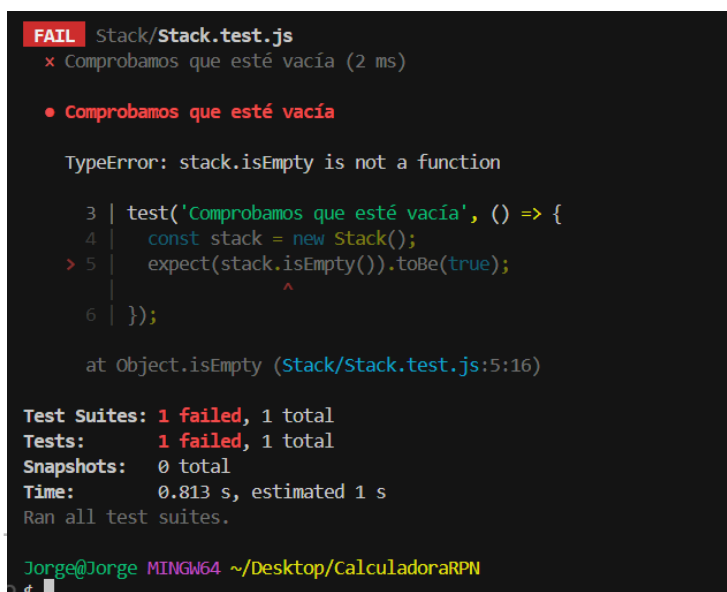
1) Comprobar que la pila esté vacía



The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows the 'Stack' folder with 'Stack.js' and 'Stack.test.js'. The main editor shows the content of 'Stack.test.js' with the following code:

```
Stack > JS Stack.test.js > ...
1 import Stack from './Stack';
2
3 test('Comprobamos que esté vacía', () => {
4   const stack = new Stack();
5   expect(stack.isEmpty()).toBe(true);
6 });
```

Esta primera prueba debería fallar, ya que no se ha definido el contenido de la clase aún. Es sólo para asegurarnos que no es un falso positivo. Ejecutamos y obtenemos:



The screenshot shows the output of the Jest test runner. It indicates that the test 'Comprobamos que esté vacía' failed. The error message is 'TypeError: stack.isEmpty is not a function'. The test suite summary shows 1 failed test out of 1 total. The output is as follows:

```
FAIL Stack/Stack.test.js
  x Comprobamos que esté vacía (2 ms)

  ● Comprobamos que esté vacía

    TypeError: stack.isEmpty is not a function

       3 | test('Comprobamos que esté vacía', () => {
       4 |   const stack = new Stack();
    >    5 |   expect(stack.isEmpty()).toBe(true);
         |           ^
       6 | });
       |
      at Object.isEmpty (Stack/Stack.test.js:5:16)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        0.813 s, estimated 1 s
Ran all test suites.
```

Como se puede apreciar la prueba falla, por lo que nos aseguramos de que se ejecute correctamente. A continuación debemos implementar el mínimo código necesario para pasar esta prueba.

Implementamos el mínimo código necesario para pasar la prueba anterior, en este caso definimos el constructor y un método que comprueba si el stack está vacío de la manera más simple posible.

```
JS Stack.js  X  JS Stack.test.js
Stack > JS Stack.js > ...
1  class Stack {
2
3      constructor() {
4          this.items = [];
5      }
6
7      isEmpty() {
8          return this.items.length === 0;
9      }
10 }
11
12
13 export default Stack;
14
15

Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN
• $ npm test

> calculadorarpn@1.0.0 test
> jest

PASS Stack/Stack.test.js
  ✓ Comprobamos que esté vacía (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.89 s, estimated 1 s
Ran all test suites.

Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN
o $
```

Comprobamos que pasa la prueba. A partir de aquí podemos ir agregando más métodos a la clase Stack, junto a su respectiva prueba e ir comprobando en cada una el código mínimo necesario para que sean pasadas. A continuación se detallan todas las realizadas para la clase Stack:

2) Comprobar que la pila no esté vacía:

```
8  test('comprobar que la pila no esté vacía', () => {
9      const stack = new Stack();
10     stack.push(1);
11     expect(stack.isEmpty()).toBe(false);
12 });
13
```

```
FAIL Stack/Stack.test.js (6.935 s)
  ✓ Comprobamos que esté vacía (2 ms)
  ✗ comprobar que la pila no esté vacía (1 ms)

  • comprobar que la pila no esté vacía

    TypeError: stack.push is not a function

      8 | test('comprobar que la pila no esté vacía', () => {
      9 |     const stack = new Stack();
    > 10 |     stack.push(1);
        |           ^
      11 |     expect(stack.isEmpty()).toBe(false);
      12 | });
      13 |

    at Object.push (Stack/Stack.test.js:10:9)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        7.714 s
Ran all test suites.
```

Ejecutamos la prueba. Al igual que la anterior fallará porque el código necesario aún no está implementado.

Agregamos el mínimo código necesario para pasar la prueba. En este caso implementamos el método `push`, que servirá para agregar un elemento al stack (usando el método `push` de la clase `Array`).

```
push(element) {  
  this.items.push(element);  
}
```

Comprobamos que pase la prueba, y cómo se aprecia a continuación la prueba pasa con éxito.

```
Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN  
$ npm test  
  
> calculadorarpn@1.0.0 test  
> jest  
  
PASS Stack/Stack.test.js  
  ✓ Comprobamos que esté vacía (2 ms)  
  ✓ comprobar que la pila no esté vacía (1 ms)  
  
Test Suites: 1 passed, 1 total  
Tests:      2 passed, 2 total  
Snapshots:  0 total  
Time:       0.953 s, estimated 7 s  
Ran all test suites.
```

El método será el mismo para todos los casos, por lo que de ahora en adelante sólo se mostrará cuál es la prueba y el código mínimo necesario para pasarla.

3) Comprobar el último elemento agregado al stack:

Test:

```
13
14 test('Devolver el último elemento agregado al Stack', () => {
15   const stack = new Stack();
16   stack.push(10);
17   expect(stack.checkLast()).toBe(10);
18   expect(stack.isEmpty()).toBe(false);
19 });
20
```

Ejecución del test

```
Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN
$ npm test

> calculadorarpn@1.0.0 test
> jest

FAIL Stack/Stack.test.js
  ✓ Comprobamos que esté vacía (2 ms)
  ✓ comprobar que la pila no esté vacía
  ✗ Devolver el último elemento agregado al Stack (2 ms)

  ● Devolver el último elemento agregado al Stack

    expect(received).toBe(expected) // Object.is equality

    Expected: 10
    Received: undefined

    15 |   const stack = new Stack();
    16 |   stack.push(10);
    17 |   expect(stack.checkLast()).toBe(10);
      |                               ^
    18 |   expect(stack.isEmpty()).toBe(false);
    19 | });
    20 |

    at Object.toBe (Stack/Stack.test.js:17:29)
```

Código mínimo:

```
14 ✓ checkLast() {
15   console.log(this.items[this.items.length - 1]);
16   return this.items[this.items.length - 1];
17 }
18 }
```

Aprobación del test:

```
Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN
$ npm test

> calculadorarpn@1.0.0 test
> jest

  console.log
    10

    at Stack.log [as checkLast] (Stack/Stack.js:15:13)

PASS Stack/Stack.test.js
  ✓ Comprobamos que esté vacía (2 ms)
  ✓ comprobar que la pila no esté vacía
  ✓ Devolver el último elemento agregado al Stack (15 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.877 s, estimated 1 s
Ran all test suites.
```

4) Eliminar el último elemento agregado a la pila

Test:

```
20 |  
21 | test('Eliminar el último elemento agregado', () => {  
22 |   const stack = new Stack();  
23 |   stack.push(1);  
24 |   stack.push(2);  
25 |   expect(stack.pop()).toBe(2);  
26 | });  
27 |
```

Ejecución del test:

```
✓ Devolver el último elemento agregado al Stack  
✗ Eliminar el último elemento agregado  
  
● Eliminar el último elemento agregado  
  
TypeError: stack.pop is not a function  
  
23 |   stack.push(1);  
24 |   stack.push(2);  
25 |   expect(stack.pop()).toBe(2);  
26 | };  
27 |  
at Object.pop (Stack/Stack.test.js:25:16)
```

Código mínimo:

```
19 | pop() {  
20 |   return this.items.pop();  
21 | }  
22 |
```

Aprobación del test:

```
Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN  
● $ npm test  
  
> calculadorarpn@1.0.0 test  
> jest  
  
PASS Stack/Stack.test.js  
  ✓ Comprobamos que la pila esté vacía (2 ms)  
  ✓ comprobar que la pila NO esté vacía (1 ms)  
  ✓ Devolver el último elemento agregado al Stack (1 ms)  
  ✓ Eliminar el último elemento agregado  
  
Test Suites: 1 passed, 1 total  
Tests: 4 passed, 4 total  
Snapshots: 0 total  
Time: 0.766 s, estimated 1 s  
Ran all test suites.
```

5) Vaciar la pila

Test:

```
28 test('Vaciar la pila', () => {
29   const stack = new Stack();
30   stack.push(1);
31   stack.push(2);
32   stack.clear();
33   expect(stack.isEmpty()).toBe(true);
34 });
35
```

Código mínimo:

```
23 clear() {
24   this.items = [];
25 }
```

Aprobación del test:

```
Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN
● $ npm test

> calculadorarpn@1.0.0 test
> jest

PASS Stack/Stack.test.js
  ✓ Comprobamos que la pila esté vacía (3 ms)
  ✓ comprobar que la pila NO esté vacía
  ✓ Devolver el último elemento agregado al Stack
  ✓ Eliminar el último elemento agregado
  ✓ Vaciar la pila

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.863 s, estimated 1 s
Ran all test suites.
```

Clase CalculadoraRPN

Prueba de suma:

Test:

```
JS Stack.js    JS CalculadoraRPN.js    JS CalculadoraRPN.test.js X
Calculadora > JS CalculadoraRPN.test.js > ...
1  import RPNCalculator from './CalculadoraRPN';
2
3  test('Prueba de suma "3 2 +"', () => {
4    const calculator = new RPNCalculator();
5    expect(calculator.evaluate("3 2 +")).toBe(5);
6  });
```

Ejecución del test:

```
PASS Stack/Stack.test.js
FAIL Calculadora/CalculadoraRPN.test.js
  ● Prueba de suma "3 2 +"

    TypeError: calculator.evaluate is not a function

      3 | test('Prueba de suma "3 2 +"', () => {
      4 |   const calculator = new RPNCalculator();
    > 5 |   expect(calculator.evaluate("3 2 +")).toBe(5);
        |                        ^
      6 | });
      |
      at Object.evaluate (Calculadora/CalculadoraRPN.test.js:5:21)

Test Suites: 1 failed, 1 passed, 2 total
Tests:       1 failed, 5 passed, 6 total
Snapshots:   0 total
Time:        1.09 s
Ran all test suites.
```

Código mínimo:

```
evaluate(operacion) {
  const stack = [];
  const tokens = operacion.split(' ');

  for (let token of tokens) {
    if (!isNaN(token)) {
      stack.push(Number(token));
    } else {
      const b = stack.pop();
      const a = stack.pop();
      if (token === '+') {
        stack.push(a + b);
      }
    }
  }

  return stack[0];
}
```

Aprobación del test:

```
Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN
● $ npm test

> calculadorarpn@1.0.0 test
> jest

PASS Stack/Stack.test.js
PASS Calculadora/CalculadoraRPN.test.js

Test Suites: 2 passed, 2 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        0.971 s, estimated 1 s
Ran all test suites.
```

Test:

```
8  ✓ test('Prueba de resta expresión "3 2 -"', () => {
9    |   const calculator = new RPNCalculator();
10   |   expect(calculator.evaluate("3 2 -")).toBe(1);
11   | });
12
13
```



```
Jorge@Jorge MINGW64 ~/Desktop/CalculadoraRPN
❶ $ npm test

> calculadorarpn@1.0.0 test
> jest

FAIL Calculadora/CalculadoraRPN.test.js
  ● Prueba de resta expresión "3 2 -"

    expect(received).toBe(expected) // Object.is equality

    Expected: 1
    Received: undefined

       8 |   test('Prueba de resta expresión "3 2 -"', () => {
       9 |     const calculator = new RPNCalculator();
    >  10 |     expect(calculator.evaluate("3 2 -")).toBe(1);
         |                                         ^
       11 |   });
       12 |
       13 |

      at Object.toBe (Calculadora/CalculadoraRPN.test.js:10:42)
```