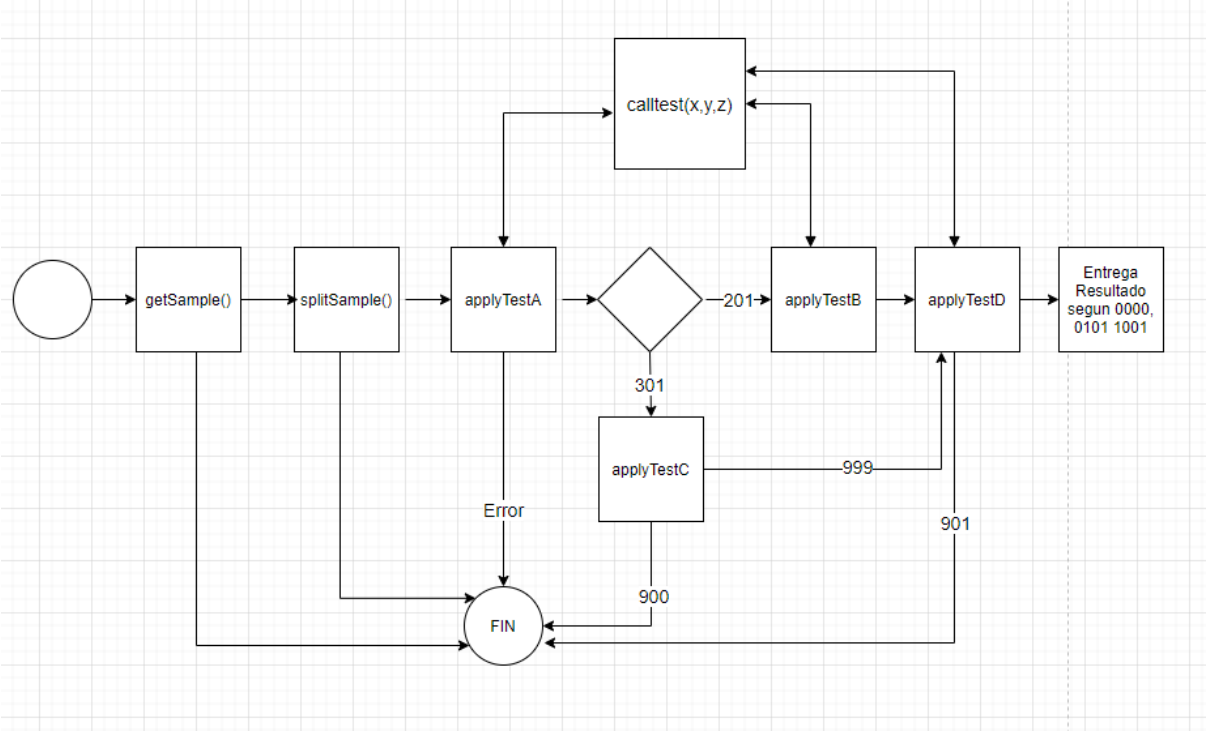
Pregunta H Interrogación IIC2513 2021-1

SECCIÓN: Programación 2

Experimentos marcianos

El 22 de mayo de 2021, la sonda China *Tianwen-1* llegó al planeta Marte y se posó en la zona conocida como Utopia Planitia. Esta sonda desplegó con éxito el rover *Zhurong,* que comenzó su exploración del planeta tomando fotografías de la superficie marciana y realizando diversos experimentos geológicos.

Se le pide a usted replicar uno de los experimentos, para lo cual deberá programar lo siguiente (el diagrama es ilustrativo de la pregunta, no contiene toda la información):



<u>Se Pide</u>:

1. Implementar las funciones **`splitSample`**, **`applyTestA`**, **`applyTestB`**, **`applyTestC`** y **`applyTestD`**. Además puede crear todas las funciones que estime necesarias.
2. Implementar la función **`performExperiment`**, que será la encargada de llamar a las funciones del punto 1 con el flujo descrito más adelante.
   Contará con las funciones asíncronas **`getSample()`** y **`callTest()`**, las cuales debe utilizar para la implementación de las funciones del punto 1. Estas dos funciones **NO pueden ser modificadas**.

3. Puede crear las estructuras de datos (objetos) que estime convenientes
4. Donde lo estime conveniente, puede usar **`Promises`** y/o **`async/await`**

A continuación encontrará el detalle del flujo de llamadas a las diferentes funciones que debe implementar.

Cada vez que el Zhurong deba recoger una muestra de roca, se invoca a la función: **`getSample()`** (el código de esta función se entrega más adelante en el enunciado, y ya se encuentra implementado en el proyecto base), la cual, cuando ha concluido el proceso mecánico de recopilación de la muestra, retorna un código de éxito (100). Si algo ha fallado, retorna un código 199, lo que deberá arrojar una excepción y error, suspendiendo toda la operación.

Si se recibe el código 100, se procede a dividir las muestras en 4 partes. Para hacer eso se invoca a la función **`splitSample()`**. Concluida con éxito, esta función retorna la siguiente estructura (los valores son de ejemplo):

```
sampleSet = [
 { sample: "A", weight:123, slot:1 },
 { sample: "B", weight:111, slot:2 },
 { sample: "C", weight:33, slot:3 },
 { sample: "D", weight:221, slot:4 },
]
```

Los valores de weight y slot son variables y en este caso sólo de ejemplo, sin embargo, **los valores de sample ("A", "B", "C", "D") no pueden cambiar pues son usados más adelante.** Al momento de implementar la función **`splitSample()`** tome en cuenta que los valores de weight serán reemplazados para poder probar su respuesta en varios escenarios de acuerdo a valores de tipo constante que se indican más adelante y que usted debe implementar también.

De fallar **`splitSample()`**, se arroja el error y se suspende la operación con el mensaje "*The sample splitting process has failed.*"

Si **`splitSample`**`()` resulta exitoso, entonces se invoca a la función: **`applyTestA()`** que busca en **`sampleSet`** por el slot en el que se encuentra la muestra A y la recoge desde allí. Si la muestra pesa menos que **`MIN_WEIGHT_A`** (que es una constante) se retorna error, con código 901 y con el siguiente mensaje: "*The sample A weight is less than that required to run the test*" y la operación se cancela.

Si el peso es adecuado, debe mostrar el mensaje: "Applying test on sample A from slot [slot]" donde [slot] es el número de slot desde donde se recuperó la muestra.

La prueba, es decir, la función **`applyTestA()`**, debe ejecutar la función (declarada más adelante en el enunciado) llamada **`callTest(0, weight, slot)`** con los valores de `weight` y `slot`, entregados en `sampleSet`. Esta función puede retornar el código 201 que es "positive", o 301 que es "negative".

Si retorna positive (201) se deberá aplicar la siguiente prueba sobre la muestra B usando para ello la función **`applyTestB()`**. De lo contrario deberá aplicar la prueba sobre la muestra C usando la función **`applyTestC()`**.

La función **`applyTestB()`** también comprueba el peso de la muestra ”B”. Si esta muestra pesa menos de **`MIN_WEIGHT_B`** (que es una constante) se retorna error con el código 901 y con el mensaje “*The sample B weight is less than that required to run the test*”. La función, en caso de que la muestra pese lo adecuado, debe invocar a **`callTest(1, weight, slot)`** que puede retornar dos valores: “type-H” o “type-G”

La función **`applyTestC()`** no comprueba el peso de la muestra ”C”. En cambio, lo que hace es ejecutar **`callTest(2, weight, slot)`**. Si el valor retornado es el código 201 (positive), **`applyTestC`** retorna el código 999 y el mensaje “Falso Negativo”. De lo contrario, si luego de ejecutar el **`callTest`** retorna el código 301 (negative), **`applyTestC()`** termina con el código 900 y el mensaje “Muestras negativas”.

Ya sea que **`applyTestC()`** retorne 999 o **`applyTestB()`** se haya aplicado con éxito, se ejecuta la prueba D con **`applyTestD()`**. Se comprueba el peso contra **`MIN_WEIGHT_D`**. Si la muestra es menor al peso requerido, se retorna código de error 901 y el mensaje “*The sample D weight is less than that required to run the test*”. Si el peso es adecuado, se aplica el test, invocando a la funcion callTest(3, weight, slot), que puede retornar los códigos 0000, 0101 o 1001.

Para finalizar el programa, se arma un arreglo con todos los resultados, correspondientes a objetos por cada prueba identificando la prueba realizada y su código de la siguiente forma:

```
{ test: ”<letra con prueba>”, result: <codigo> }.
```

Si la prueba no se aplicó, se da como resultado “NA”. A continuación algunos ejemplos:

```
[
  { test: ”A”, result: 201 },
  { test: ”B”, result: ”type-G” },
  { test: ”C”, result: ”NA” },
  { test: ”D”, result: 901 },
]
```

o también:

```
[
  { test: ”A”, result: 301 },
  { test: ”B”, result: ”NA” },
  { test: ”C”, result: 999 },
  { test: ”D”, result: ”NA” },
]
```

otro posible caso:

```
[
  { test: ”A”, result: 901 },
  { test: ”B”, result: ”NA” },
  { test: ”C”, result: ”NA” },
  { test: ”D”, result: ”NA” },
]
```

Un último caso:

```
[
  { test: ”A”, result: 201 },
  { test: ”B”, result: ”type-G” },
  { test: ”C”, result: ”NA” },
  { test: ”D”, result: 0101 },
]
```

Dependiendo del resultado (positivo) en el test D, se debe retornar el siguiente mensaje final:

- 0000: “Not evidence found”
- 0101: “Weak evidence”
- 1001 y el testB con codigo “type-G” retorna mensaje: “Proceed with further tests in other selected areas”
- 1001 y el testB con codigo “type-H” retorna mensaje: “Positive outcome, proceed with further test on site”

Para las pruebas se DEBE usar la siguiente función:

```
function callTest( test, weight, slot )
{
   //Valores para testear
   let arrayResults = [201, 'type-H', 0, '0101'];
   return new Promise( (resolve, reject) =>{
     if (weight > 1000)
       weight = 1000;
     else
      weight = weight*10;
     let wait = weight * slot;

     setTimeout( () => resolve(arrayResults[test]),
wait)
```

```
    });
}
```

Esta función recibe un número de 0 a 3 dependiendo del tipo de test (0 para A, 1 para B y así sucesivamente).

`arrayResults` es un arreglo que servirá para testar su código, debe contener el resultado de acuerdo al tipo de test. Se puede modificar para ver que se comporta adecuadamente el código. Por ejemplo podemos tener arrayResults = [201, “type-H”, 0, “0101”] como en el código mostrado anteriormente. El propósito es poder retornar distintos valores para probar su programa.

La función retornará **`arrayResults[test]`**, es decir, un único valor que corresponderá al retorno después de haber aplicado el test correspondiente. Por ejemplo, si se ejecuta el testB, se invoca a la función como **`callTest(1, 508, 2)`** y se retornará **`arrayResults[1]`**, que en el caso de este ejemplo, corresponde a “type-H”. En este caso, el parámetro test es igual a 1 pues es el test “B”.

Para `getSample()` debe usar la siguiente función:

```
function getSample()
{
 let code = 100; //100 para exito o 199, fail
 let delay = Math.floor(Math.random() * 10);
  if (delay == 0) delay = 1;

 return new Promise( (resolve, reject) =>{

   if(code == 109) {
     reject("error 109");
   }

   setTimeout( () => resolve(code), delay * 1000 );
  });

}
```