Máquina de cambio

Seguramente te hayas encontrado en más de una tienda, una máquina en la que pagas con un billete o monedas y ésta te devuelve el cambio exacto, con los billetes y monedas más grandes posibles ¿Cómo podemos implementar esto?

Enunciado (basado en caso real)

En la panadería de tu barrio, no quieren que las mismas personas que manipulan los alimentos toquen el dinero, para ello los clientes tendrán que pagar utilizando una máquina, dicha máquina:

- Admite billetes de 5, 10, 20 y 50 €
- Admite monedas de 1, 2, 5, 10, 20, 50 céntimos, 1 y 2 €

Cuando el usuario ha introducido las monedas o billetes pertinentes, la máquina calcula el cambio y lo devuelve, dando prioridad a los billetes y monedas más grandes.

Ejemplo A:

- La compra son 2,50 €
- El usuario paga con un billete de 50 €
- La máquina devuelve 47,50 € en billetes y monedas:
    - 2 billete de 20 €
    - 1 billete d 5 €
    - 1 moneda de 2 €
    - 1 moneda de 0,50 €

Ejemplo B:

- La compra son 4,82 €
- El usuario paga 5,32 €
- La máquina devuelve 0,50 € en monedas:
    - 1 moneda de 0,50 €

Ejemplo C:

- La compra son 2 €
- El usuario paga 6 €
- La máquina devuelve 4 € en monedas:
    - 2 monedas de 2 €

¿Cómo se puede resolver esto?

- Primero tengo un array de billetes y monedas ordenados de mayor a menor.
- Empiezo por el billete más grande: divido la cantidad que tengo por el billete:

    - Si me da mayor que cero ya sé que para el billete cuantos tengo que devolver (si el cociente es cero quiere decir que el billete es más grande que la cantidad a pagar).
    - Por otro lado, el resto me dice cuanta cantidad me queda por devolver (si no es cero, necesito billetes o monedas más pequeños), esta cantidad de resto me sirve para:
        - Si el resto es cero parar con el algoritmo (ya tengo el cambio calculado).
        - Sino:
            - Avanzar un billete en el array (voy al siguiente billete o moneda más pequeño).
            - Y Volver al paso inicial (dividir la cantidad que me queda por el billete actual), así hasta que resto sea cero.

Firma del método

Parámetros de entrada:

En este caso vamos a tener dos parámetros de entrada:

- El importe de la compra.
- La cantidad que paga el cliente.

Parámetro de salida:

Aquí vamos a crear una interfaz para definir el tipo de objeto que vamos a devolver.

```
interface Cambio {
  moneda: number;
  cuantos: number;
}
```

Nos devolvería un array de objetos, cada objeto tendrá dos propiedades, la *moneda/billete* y cuantos:

```
[{moneda: 5, cuantos: 2}, {moneda: 2, cuantos: 1}]
```

La firma se podría quedar de la siguiente manera:

```
const calcularCambio = (compra: number, pago: number): Cambio[] => {
  // TODO
};
```

Evaluando el problema

Vamos a dividir el problema en partes más pequeñas:

1. Tengo un array de billetes y monedas ordenados de mayor a menor.

2. Calculo una sola entrada, devolver un objeto que indica cuantos de ese billete o moneda tengo que devolver y cuanto me queda pendiente de calcular de vuelta en monedas o billetes más pequeños.
3. Calculo el cambio completo, devolver un array de objetos con el cambio.

Implementación

Creando el array de billetes y monedas

```
const arrayBilletesMonedas = [
  50, 20, 10, 5, 2, 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01,
];
```

Calculando una sola entrada

Vamos a crear una función que reciba la cantidad a devolver y un billete o moneda y nos devuelva un objeto con la cantidad de ese billete o moneda y el resto que me queda por repartir en monedas más pequeñas.

Vamos a crear un fichero que vamos a llamar *./src/cambio.helper.ts*

La función se podría llamar:

```
interface Resultado {
  cuantos: number;
  restoCantidad: number;
}

const calcularEntrada = (
  cantidad: number,
  billeteMoneda: number
): Resultado => {
  // TODO
};
```

Vamos a añadir unas pruebas para ver que funciona correctamente, te toca intenta añadir los siguientes casos de prueba:

- Debería devolver un throw si los parámetros de entrada son undefined
- Debería devolver un throw si los parámetros de entradas son null
- Si tengo que devolver 2.5 € y quiero ver cuantos billetes de 50, tendría que:
    - Decirme que tengo que devolver 0 billetes de 50
    - Decirme que me queda 2.5 € por devolver
- Si tengo que devolver 2.25 € y quiero ver cuantas monedas de 2, tendría que::

    - Decirme que tengo que devolver 1 moneda de 2
    - Decirme que me queda 0.5 € por devolver

- Si tengo que devolver 7.25 € y quiero ver cuantos billetes de 5, tendría que:

    - Decirme que tengo que devolver 1 billete de 5
    - Decirme que me queda 2.5 € por devolver

- Aquí podría jugar con varios casos más

Otro caso arista interesante sería que *billeteMoneda* sea cero, si quieres pruébalo y mira que pasa 😃.

Creamos un fichero que llamaremos *./src/cambio.helper.spec.ts* y añadimos las pruebas:

```
describe("calcularEntrada", () => {
  it("debería devolver un throw si las entradas son undefined", () => {
    // Arrange
    const cantidad: any = undefined;
    const moneda: any = undefined;

    // Act
    const result = () => calcularEntrada(cantidad, moneda);

    // Assert
    expect(result).toThrowError("Los parámetros introducidos no son correctos");
  });

  it("debería devolver un throw si las entradas son null", () => {
    // Arrange
    const cantidad: any = null;
    const billeteMoneda: any = null;

    // Act
    const result = () => calcularEntrada(cantidad, billeteMoneda);

    // Assert
    expect(result).toThrowError("Los parámetros introducidos no son correctos");
  });

  it("Devolver: 2.5, billete 50 --> {cuantos: 0, restoCantidad: 2.5}", () => {
    // Arrange
    const cantidad = 2.5;
    const billeteMoneda = 50;

    // Act
    const result = calcularEntrada(cantidad, billeteMoneda);

    // Assert
    const expected = { cuantos: 0, restoCantidad: 2.5 };
    expect(result).toEqual(expected);
  });

  it("Devolver: 2.5, billete 2 --> {cuantos: 1, restoCantidad: 0.5}", () => {
```

```
    // Arrange
    const cantidad = 2.5;
    const billeteMoneda = 2;

    // Act
    const result = calcularEntrada(cantidad, billeteMoneda);

    // Assert
    const expected = { cuantos: 1, restoCantidad: 0.5 };
    expect(result).toEqual(expected);
  });

  it("Devolver: 7.25, billete 5 --> {cuantos: 1, restoCantidad: 2.25}", () => {
    // Arrange
    const cantidad = 7.25;
    const billeteMoneda = 5;

    // Act
    const result = calcularEntrada(cantidad, billeteMoneda);

    // Assert
    const expected = { cuantos: 1, restoCantidad: 2.25 };
    expect(result).toEqual(expected);
  });
});
```

Te toca, intenta implementarlo.

Pistas:

- Para calcular la cantidad de billetes o monedas que tengo que devolver, puedes utilizar la función Math.floor() que te devuelve la parte entera de un número.
- Para calcular el resto de la cantidad que me queda por devolver, puedes utilizar el operador %.
- Devuelve un objeto con las propiedades cuantos y restoCantidad.

A continuación, vamos a implementar la función:

```
export const calcularEntrada = (
  cantidad: number,
  billeteMoneda: number
): Resultado => {
  if (!cantidad || !billeteMoneda) {
    throw new Error("Los parámetros introducidos no son correctos");
  }
  const cuantos = Math.floor(cantidad / billeteMoneda);
  const restoCantidad = cantidad % billeteMoneda;
  return { cuantos, restoCantidad };
};
```

Esta función nos devuelve un objeto con la cantidad de billetes o monedas que tenemos que devolver y el resto de la cantidad que nos queda por devolver.

Calculando el cambio completo

Una vez que hemos calculado una sola entrada, es la hora de calcular el cambio completo, para ello tenemos la función que recibe la cantidad de la compra y la cantidad que paga el cliente y nos devuelva un array de objetos con el cambio.

La función se podría llamar:

```
const calcularCambio = (compra: number, pago: number): Cambio[] => {
  // TODO
};
```

Vamos a añadir unas pruebas para ver que funciona correctamente...

Bueno, te toca 😃, implemnta los siguientes casos de prueba:

- Throw sin parametros de entrada son undefined.
- Throw sin parametros de entrada son null.
- Compra 2.5 €, usuario paga 50 € --> devolucion [2 de 20, 1 de 5, 2 de 1, 1 de 0.5]
- Compra 4.82 €, usuario paga 5.32 € --> devolucion [1 moneda de 0.5]"
- Compra 2 €, usuario paga 6 € --> devolucion [2 monedas de 2]

```
describe("calcularCambio", () => {
  it("debería devolver un throw si las entradas son undefined", () => {
    // Arrange
    const compra: any = undefined;
    const pago: any = undefined;

    // Act
    const result = () => calcularCambio(compra, pago);

    // Assert
    expect(result).toThrowError("Los parámetros introducidos no son correctos");
  });

  it("debería devolver un throw si las entradas son null", () => {
    // Arrange
    const compra: any = null;
    const pago: any = null;

    // Act
    const result = () => calcularCambio(compra, pago);

    // Assert
    expect(result).toThrowError("Los parámetros introducidos no son correctos");
  });
```

```
  it("2.5 €, usuario paga 50 € --> devolucion [2 de 20, 1 de 5, 2 de 1, 1 de
0.5]", () => {
    // Arrange
    const compra = 2.5;
    const pago = 50;

    // Act
    const result = calcularCambio(compra, pago);

    // Assert
    const expected: Cambio[] = [
      { moneda: 20, cuantos: 2 },
      { moneda: 5, cuantos: 1 },
      { moneda: 2, cuantos: 1 },
      { moneda: 0.5, cuantos: 1 },
    ];
    expect(result).toEqual(expected);
  });

  it("4.82 €, usuario paga 5.32 € --> devolucion [1 moneda de 0.5]", () => {
    // Arrange
    const compra = 4.82;
    const pago = 5.32;

    // Act
    const result = calcularCambio(compra, pago);

    // Assert
    const expected: Cambio[] = [{ moneda: 0.5, cuantos: 1 }];
    expect(result).toEqual(expected);
  });

  it("2 €, usuario paga 6 € --> devolucion [2 monedas de 2]", () => {
    // Arrange
    const compra = 2;
    const pago = 6;

    // Act
    const result = calcularCambio(compra, pago);

    // Assert
    const expected: Cambio[] = [{ moneda: 2, cuantos: 2 }];
    expect(result).toEqual(expected);
  });
});
```

Te toca, intenta implementarlo.

Pistas:

- Para calcular el cambio completo, puedes utilizar la función reduce de los *array methods*.

- Para calcular una sola entrada, puedes utilizar la función calcularEntrada que hemos creado anteriormente.
- Devuelve un array de objetos con las propiedades moneda y cuantos, puedes tipar la salida de la función con la interfaz *Cambio* que creamos anteriormente.

Ahora ya podemos implementar la función:

```
export const calcularCambio = (compra: number, pago: number): Cambio[] => {
  if (!compra || !pago) {
    throw new Error("Los parámetros introducidos no son correctos");
  }
  const cambio = pago - compra;
  let cambioRestante = cambio;

  const resultado: Cambio[] = arrayBilletesMonedas.reduce(
    (acumulador: Cambio[], billeteMoneda: number) => {
      if (cambioRestante === 0) {
        return acumulador;
      }

      const { cuantos, restoCantidad } = calcularEntrada(
        cambioRestante,
        billeteMoneda
      );
      cambioRestante = restoCantidad;

      return cuantos > 0
        ? [...acumulador, { moneda: billeteMoneda, cuantos }]
        : acumulador;
    },
    []
  );

  return resultado;
};
```