The Chocolate factory

Introduction

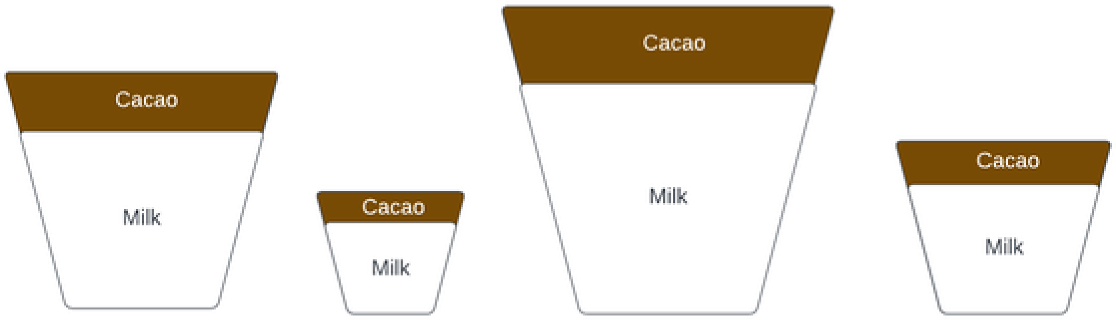
Complete and submit the following exercise.

You can solve this exercise using the programming language you prefer, however the main file and the boilerplate class are provided in Javascript (suggested). If you don't use Javascript, you will need to rewrite the **main** file and the **Machine** Class.

Exercise

A factory of ice cream produces a single taste of ice cream (chocolate).

In order to do so, they use a machine that fills variable-capacity *buckets* with 20% of cacao and 80% of milk. Refer to the following image for a visual representation of the buckets:



A *bucket* is an object-value represented in the following way:

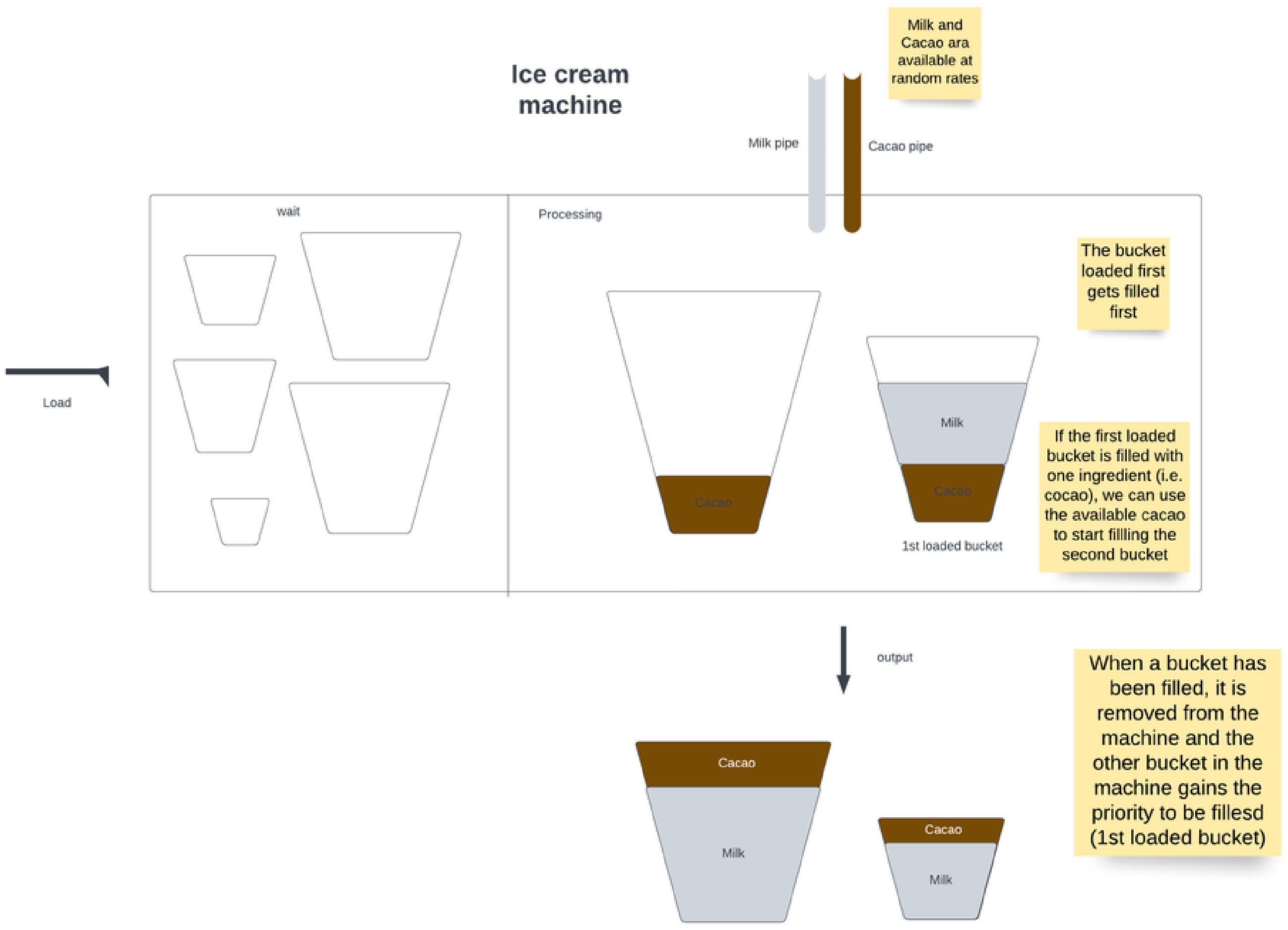
```
{
        capacity: 10,
        milk: 0,
        cacao: 0
}
```

Where *capacity* is the total capacity of the bucket, *milk* is the amount of milk and *cacao* is the amount of cacao in the bucket. *cacao + milk <= capacity*.

The **Machine** producing the ice cream has the following specification:

- The ***Machine*** processes 2 *buckets* per time (2 slots available). More than 2 *buckets* can be loaded on the machine at the same time (with the `load()` method) but only 2 can be processed simultaneously. Others will stay in a "wait state" until a slot gets free (i.e. one of the *buckets* being processed is filled and unloaded);
- When one of the two processing *buckets* has been filled, it is removed from the processing state and a new slot is made available for a new empty *bucket*
- The supply of milk and cacao to the **machine** happens through a pipe at random rate. *Buckets* stay in processing state until **all** the required milk (80%) and cacao (20%) has been put into the bucket
- The available milk or cacao are always added first to the least recently added *bucket*. When starting (i.e. two buckets are loaded simultaneously), the first bucket in the list has priority.
- Be careful: if the first loaded *bucket* is fully filled with an ingredient (i.e. 20% of it is full of cacao), the available ingredient (i.e. cacao) must be used to start filling the second bucket.
- A *bucket* is considered filled when 20% cacao + 80% milk == `capacity`. When a *bucket* has been filled, it is unloaded from the **machine** and a new empty *bucket* starts processing. The more recently added *bucket* which was in processing state gains priority.
- The **machine** continues working until all the provided buckets (with the `load` method) are filled.

Refer to the following image for a visual representation of the **Machine:**


Ice cream machine
Milk and Cacao ara available at random rates
Milk pipe
Cacao pipe
wait
Processing
Load
Cacao
Milk
Cacao
1st loaded bucket
The bucket loaded first gets filled first
If the first loaded bucket is filled with one ingredient (i.e. cocao), we can use the available cacao to start fillling the second bucket
output
Cacao
Milk
Cacao
Milk
When a bucket has been filled, it is removed from the machine and the other bucket in the machine gains the priority to be fillesd (1st loaded bucket)

Model the above scenario by providing an implementation for the Machine class (in Javascript):

```
class Machine {
        constructor() {

        }

        addMilk(amount) {

        }

        addCacao(amount) {

        }

        async load(buckets, onBucketReady) {

        }
}
```

You can add additional methods but not changing the provided signatures' methods.

- The `addMilk` method add a variable amount of milk to the machine
- The `addCacao` method add a variable amount of cacao to the machine

The `load` method loads all the *buckets* into the **Machine** and returns a promise which is resolved only when **ALL** the *buckets* have been filled, and it takes a callback as second parameter (the implementation is provided) that is called each time a *bucket* is filled with the *bucket* object as parameter

The programs needs to run using Node.js or as a browser script. Don't use any external library. If you choose another programming language, provide instructions for running the program.

Main

Having this main program (in Javascript) which is using your **Machine** implementation:

```
(async () => {
        const buckets = [{
                capacity: 10,
                milk: 0,
                cacao: 0
        }, {
                capacity: 1000,
                milk: 0,
                cacao: 0
        }, {
                capacity: 500,
                milk: 0,
                cacao: 0
        }, {
                capacity: 4000,
                milk: 0,
                cacao: 0
        }, {
                capacity: 200,
                milk: 0,
                cacao: 0
        }, {
                capacity: 1000,
                milk: 0,
                cacao: 0
        }, {
                capacity: 300,
                milk: 0,
                cacao: 0
        }, {
                capacity: 0,
                milk: 0,
                cacao: 0
        }]

        const machine = new Machine()

        const job = setInterval(() => {
                machine.addMilk(Math.random() * 100)
                machine.addCacao(Math.random() * 100)
        }, Math.random() * 100)

        await machine.load(buckets, onBucketReady)

        clearInterval(job)

        console.log('Finished filling all the buckets')
})()
```

if your implementation of the **Machine** class is correct, the above program will print:

```
Bucket has been filled, capacity:  10 milk 8 cacao 2
Bucket has been filled, capacity:  500 milk 400 cacao 100
Bucket has been filled, capacity:  1000 milk 800 cacao 200
Bucket has been filled, capacity:  200 milk 160 cacao 40
Bucket has been filled, capacity:  1000 milk 800 cacao 200
Bucket has been filled, capacity:  300 milk 240 cacao 60
Bucket has been filled, capacity:  0 milk 0 cacao 0
Bucket has been filled, capacity:  4000 milk 3200 cacao 800
Finished filling all the buckets
```