Create analytics SDK in JavaScript

Problem Statement -

Implement an analytics SDK that exposes log events, it takes in events and queues them and then starts sending the events.

- Send each event after a delay of 1 second and this logging fails every n % 5 times.
- Send the next event only after the previous one resolves.
- When the failure occurs, attempt a retry.

Example

```
Input:
const sdk = new SDK();

sdk.logEvent("event 1");
sdk.logEvent("event 2");
sdk.logEvent("event 3");
sdk.logEvent("event 4");
sdk.logEvent("event 5");
sdk.logEvent("event 6");
sdk.logEvent("event 7");
sdk.logEvent("event 8");
sdk.logEvent("event 9");
sdk.logEvent("event 10");

sdk.send();

Output:
"Analytics sent event 1"
"Analytics sent event 2"
"Analytics sent event 3"
"Analytics sent event 4"
"-----------------------"
"Failed to send event 5"
"Retrying sending event 5"
"-----------------------"
"Analytics sent event 5"
"Analytics sent event 6"
```

```
"Analytics sent event 7"
"Analytics sent event 8"
"-----------------------"
"Failed to send event 9"
"Retrying sending event 9"
"-----------------------"
"Analytics sent event 9"
"Analytics sent event 10"
```

Breaking the problem statement into subproblems we can create this SDK in three steps.

Delay function

The most important part of this function is that the events will be sent after a delay of 1 second and fails n%5 times.

We can do the same by extending the sleep function.

Create a new Promise and inside that run a setTimeout that will resolve the promise after the delay.

To the same, add one extra condition that will check if the current execution is n%5 then reject, else resolve.

```
  // function to delay the execution
  wait = () => new Promise((resolve, reject) => {
    setTimeout(() => {
      // reject every n % 5 time
      if(this.count % 5 === 0){
        reject();
      } else {
        resolve();
      }
    }, 1000);
  });
```

Queue the events

We have to store the events so that they can be sent one by one. We also need a tracker so that we can reject for each n%5 th call.

We can create a class and initialize these in the constructor.

```
class SDK {
  constructor(){
    // hold the events
    this.queue = [];

    // track the count
    this.count = 1;
  }

  // push event in the queue
  logEvent(ev) {
    this.queue.push(ev);
  }
}
```

Sending the events

The final part is sending the events, for this, we can create a helper function that will recursively call itself and keep on sending one-one events every time.

This will be an async function and in each call, get the first element from the queue and try the wait(), if it resolves then print the log or perform any other operation, else if it fails, push the event back in the queue for retry. Finally, recursively call the same function for the next operation.

Add a base case to stop the execution if there are no more events in the queue. Also, track the count in each call.

```
  // to send analytics
  // recursively send the events
```

```
sendAnalytics = async function (){
  // if there are no events in the queue
  // stop execution
  if(this.queue.length === 0){
    return;
  }

  // get the first element from the queue
  const current = this.queue.shift();

  try {
    // delay
    await this.wait();

    // print the event
    // can perform any other operations as well like making api call
    console.log("Analytics sent " + current);

    // increase the count
    this.count++;
  } catch(e){

    // if execution fails
    console.log("-----------------------");
    console.log("Failed to send " + current);
    console.log("Retrying sending " + current);
    console.log("-----------------------");

    // reset the count
    this.count = 1;

    // push the event back into the queue
    this.queue.unshift(current);
  }finally{

    // recursively call the same function
    // to send the remaining
    this.sendAnalytics();
  }
}

// start the execution
send = async function(){
  this.sendAnalytics();
}
```

Putting everything together

```
class SDK {
  constructor(){
    // hold the events
    this.queue = [];

    // track the count
    this.count = 1;
  }

  // push event in the queue
  logEvent(ev) {
    this.queue.push(ev);
  }

  // function to delay the execution
  wait = () => new Promise((resolve, reject) => {
    setTimeout(() => {
      // reject every n % 5 time
      if(this.count % 5 === 0){
        reject();
      } else {
        resolve();
      }
    }, 1000);
  });

  // to send analytics
  // recursively send the events
  sendAnalytics = async function (){
    // if there are no events in the queue
    // stop execution
    if(this.queue.length === 0){
      return;
    }

    // get the first element from the queue
    const current = this.queue.shift();
    try {
      // delay
      await this.wait();

      // print the event
      // can perform any other operations as well like making api call
      console.log("Analytics sent " + current);

      // increase the count
      this.count++;
    } catch(e){

      // if execution fails
      console.log("-----------------------");
      console.log("Failed to send " + current);
      console.log("Retrying sending " + current);
      console.log("-----------------------");

      // reset the count
      this.count = 1;

      // push the event back into the queue
      this.queue.unshift(current);
    }finally{

      // recursively call the same function
      // to send the remaining
      this.sendAnalytics();
    }
  }

  // start the execution
  send = async function(){
    this.sendAnalytics();
  }
}
```

Test Case

```
Input:
const sdk = new SDK();
```

```
sdk.logEvent("event 1");
sdk.logEvent("event 2");
sdk.logEvent("event 3");
sdk.logEvent("event 4");
sdk.logEvent("event 5");
sdk.logEvent("event 6");
sdk.logEvent("event 7");
sdk.logEvent("event 8");
sdk.logEvent("event 9");
sdk.logEvent("event 10");
sdk.logEvent("event 11");
sdk.logEvent("event 12");
sdk.logEvent("event 13");
sdk.logEvent("event 14");
sdk.logEvent("event 15");
sdk.logEvent("event 16");
sdk.logEvent("event 17");
sdk.logEvent("event 18");
sdk.logEvent("event 19");
sdk.logEvent("event 20");

sdk.send();

Output:
"Analytics sent event 1"
"Analytics sent event 2"
"Analytics sent event 3"
"Analytics sent event 4"
"-----------------------"
"Failed to send event 5"
"Retrying sending event 5"
"-----------------------"

"Analytics sent event 5"
"Analytics sent event 6"
"Analytics sent event 7"
"Analytics sent event 8"
"-----------------------"
"Failed to send event 9"
"Retrying sending event 9"
"-----------------------"

"Analytics sent event 9"
```

```
"Analytics sent event 10"
"Analytics sent event 11"
"Analytics sent event 12"
"-----------------------"
"Failed to send event 13"
"Retrying sending event 13"
"-----------------------"

"Analytics sent event 13"
"Analytics sent event 14"
"Analytics sent event 15"
"Analytics sent event 16"
"-----------------------"
"Failed to send event 17"
"Retrying sending event 17"
"-----------------------"

"Analytics sent event 17"
"Analytics sent event 18"
"Analytics sent event 19"
"Analytics sent event 20"
```