Hack your future BE

JAVASCRIPT 3

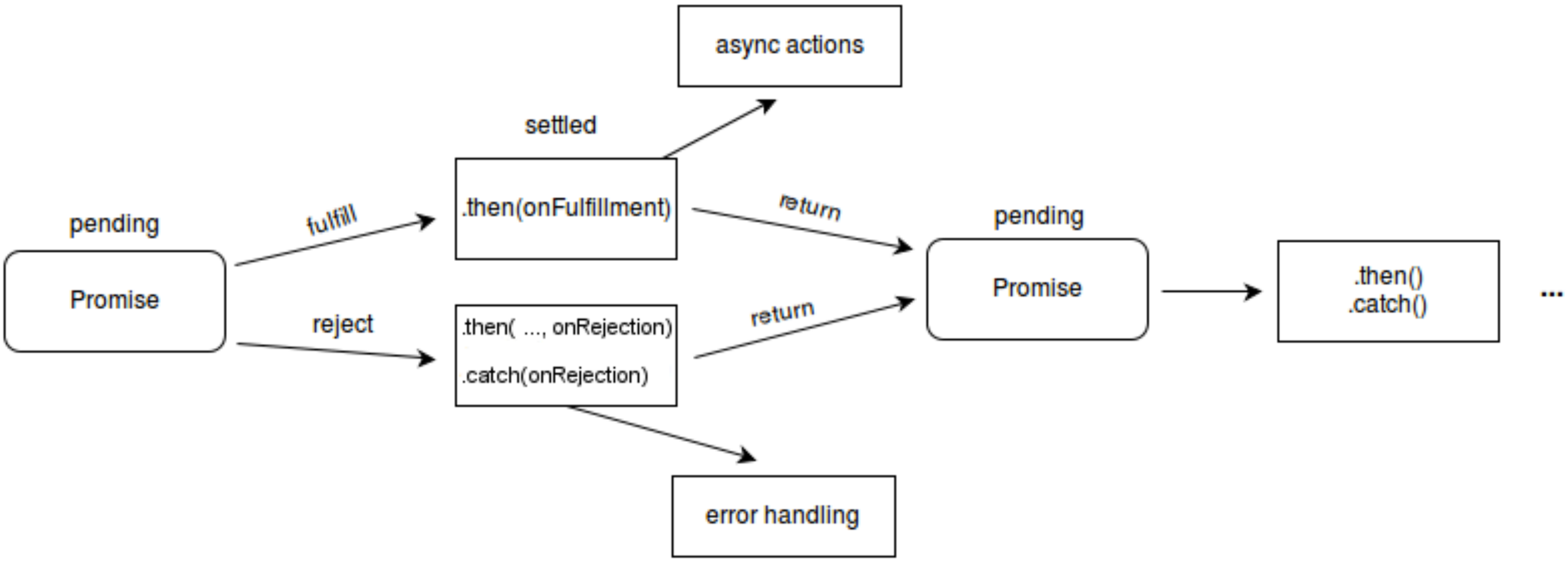
I hate programming
I hate programming
I hate programming

it works

I love programming!

```
Const promise1 = new Promise(
 (resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});

promise1.then((value) => {
  console.log(value);
});
```


async actions
settled
pending
fulfill
.then(onFulfillment)
return
pending
Promise
.then()
.catch()
...
reject
.then(..., onRejection)
.catch(onRejection)
return
Promise
error handling

```
FETCH(
   'HTTPS://JSONPLACEHOLDER.TYPICODE.COM/TODOS/1'
)
  .THEN(RESPONSE => RESPONSE.JSON())
  .THEN(JSON => CONSOLE.LOG(JSON))
```

```
function fetchAndRender(url, otherUrl) {

  fetchJSON(url)

    .then(data => {
      renderData(data);
      return fetchJSON(otherUrl);
    })

    .then(otherData => {
      renderOtherData(otherData);
    })

    .catch(err => {
      renderError(err);
    });

}
```

```
function fetchAndRender(url, otherUrl) {

  fetchJSON(url)
    .then(data => {

      return fetchJSON(otherUrl)
        .then(otherData => {
          renderData(data);
          renderOtherData(otherData);
        });

    })
    .catch(err => {
      renderError(err);
    });

}
```

```
function fetchAndRender(url, otherUrl) {

  Promise

    .all([fetchJSON(url), fetchJSON(otherUrl)])

    .then(([data, otherData]) => {
      renderData(data);
      renderOtherData(otherData);
    })

    .catch(err => {
      renderError(err);
    });

}
```

```
async function fetchAndRender(url) {
    const data = await fetchJSON(url);
    renderData(data);
}
```

AWAIT

- Causes code execution to be suspended in a non-blocking manner
- Until the async action has completed (promise is settled)
- Awaited expression returns the fulfilled value of the promise
- Execution resumes at the point where it was left off

REWRITE USING ASYNC / AWAIT

```
function fetchAndRender(url, otherUrl) {
  fetchJSON(url)
    .then(data => {
      return fetchJSON(otherUrl)
        .then(otherData => {
          renderData(data);
          renderOtherData(otherData);
        });
    })
    .catch(err => {
      renderError(err);
    });
}
```

REWRITE USING ASYNC / AWAIT

```
async function fetchAndRender(url, otherUrl) {
    const data = await fetchJSON(url);
    const otherData = await fetchJSON(otherUrl);
    renderData(data);
    renderOtherData(otherData);
}
```

ERROR HANDLING

ERRORS
&
EXCEPTIONS

TRY { }
CATCH { }

a block of statements to try
and specifies a response
should an exception be thrown

```
async function fetchAndRender() {
  try {
    const data = await fetchJSON(url);
    const otherData = await fetchJSON(otherUrl);
    renderData(data);
    renderOtherData(otherData);
  }
  catch (err) {
    renderError(err);
  }
}
```

CATCHING ERRORS

```
try {
  const obj = JSON.parse('this is invalid JSON');
  console.log(obj);
}


catch (err) {
  console.error('An error occurred: ' + err.message);
}
```

```
const months = [
  { name: 'January', days: 31 },
  { name: 'February', days: 28 },
  { name: 'March', days: 31 },
  { name: 'April', days: 30 },
  { name: 'May', days: 31 },
  { name: 'June', days: 30 },
  { name: 'July', days: 31 },
  { name: 'August', days: 31 },
  { name: 'September', days: 30 },
  { name: 'October', days: 31 },
  { name: 'November', days: 30 },
  { name: 'December', days: 31 }
];

for (const month of months) {
  if (month.days === 31) {
    console.log(`${month.name} has ${month.days} days.`);
  }
}

months
  .filter(month => month.days === 31)
  .map(month => `${month.name} has ${month.days} days.`)
  .forEach(string => console.log(string));
```

```
function Month(name, days) {
  this.name = name;
  this.days = days;

  this.hasDays = function (days) {
    return this.days === days;
  };

  this.isLongMonth = function () {
    return this.hasDays(31);
  };

  this.toString = function () {
    return `${this.name} has ${this.days} days.`;
  };

  this.toConsole = function () {
    console.log(this.toString());
  };

}

const months = [
  new Month('January', 31),
  new Month('February', 28),
  new Month('March', 31),
  new Month('April', 30),
];
```

```
months
  .filter(month => month.isLongMonth())
  .forEach(month => month.toConsole());
```

OBJECT ORIENTED PROGRAMMING

- Adding methods to an object to operate on data contained in the object

- The object knows how to operate its data and external code neet not know anything about its internals

OBJECT ORIENTED PROGRAMMING

Each object get its own copy of the methods (`hasDays()` etc)

This takes up unnecessary memory

It would be far better if the objects could share a common set of methods

```
class Month {
  constructor(name, days) {
    this.name = name;
    this.days = days;
  }

  hasDays(days) {
    return this.days === days;
  }

  isLongMonth() {
    return this.hasDays(31);
  }

  toString() {
    return `${this.name} has ${this.days} days.`;
  }

  toConsole() {
    console.log(this.toString());
  }
}
```

the `this` keyword refers to the object that a method is called upon

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }

  getArea() {
    return this.calcArea();
  }

  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.getArea())
```

HOMEWORK