Design Patterns: Singleton

Mitch Keenan

Design Patterns: Singleton

1. What are design patterns?
2. What is the Singleton pattern?
3. When should it be used?
4. Examples

Design Patterns

> A **design pattern** is a general repeatable solution to a commonly occurring problem in software design.

– Gang of Four

Examples:

- **Singleton**
- Observer
- Factory

Singleton

The intent of the Singleton is to...

> Ensure a class only has one instance, and provide a global point of access to it.

Which also implies that **it's only created once**

When should Singleton be used?

When you need to

- Persist state and make it available throughout your codebase
- Provide access to a shared resource
- Model a **truly unique** domain class

When should Singleton *not* be used?

- Modelling domain classes which are not unique
- *Ever?*

Singletons in JavaScript - 1

```
const myObject = {
  getMessage: function () {
    return "Pssst... Object Literals are secretly singletons";
  }
}
```

Singletons in JavaScript - 2

```
function MySingleton() {
  if (MySingleton.instance) {
    return MySingleton.instance;
  }

  MySingleton.getMessage = function () {
    return "I'm a singleton!!";
  }

  MySingleton.instance = this;
}

const s1 = new MySingleton();
const s2 = new MySingleton();
console.log(s1 === s2); // true
```

demo

Singletons in TypeScript - 1

```
class MySingleton {
  private static instance: MySingleton;

  private constructor () {}

  // Only way to access an instance of the class.
  static getInstance() {
    if(!MySingleton.instance) {
      MySingleton.instance = new MySingleton();
    }

    return MySingleton.instance;
  }
}

const s1 = MySingleton.getInstance();
const s2 = MySingleton.getInstance();
console.log(s1 === s2); // true

const s3 = new MySingleton(); // Compile time error
```

Singletons in TypeScript - 2

```typescript
namespace MySingleton {
  // Any initialisation goes here

  export  getMessage() {
    return "I'm a singleton!!";
  }
}
const s3 = new MySingleton(); // Compile time error
```

Angular Demo

Demo

Singleton Pros

1. Controlled access
2. Global State

Singleton Cons

1. Hard to test
2. Global State
3. **Global State**

Resources

Demos

- Typescript Singleton Demo 1
- Typescript Singleton Demo 2
- Javascript Demo
- Angular Demo

Reading

- Rod Dodson: Singleton (*warning*: explicit language)
- Source Making: Singleton
- Addy Osmani: Singleton
- Design Patterns - Gang of Four (*warning*: **slow** pdf link)