DESIGN PATTERNS
USING JAVASCRIPT

D02

Design patterns categories

- Creational Design Pattern
- Structural Design Patterns
- Behavioral design patterns

Creational Design Patterns

As the name suggests, it provides the object or classes creation **mechanism** that enhance the **flexibilities** and **reusability** of the existing code.

- Abstract Factory
- Builder
- **Factory Method**
- **Prototype**
- **Singleton**

Structural Design Patterns

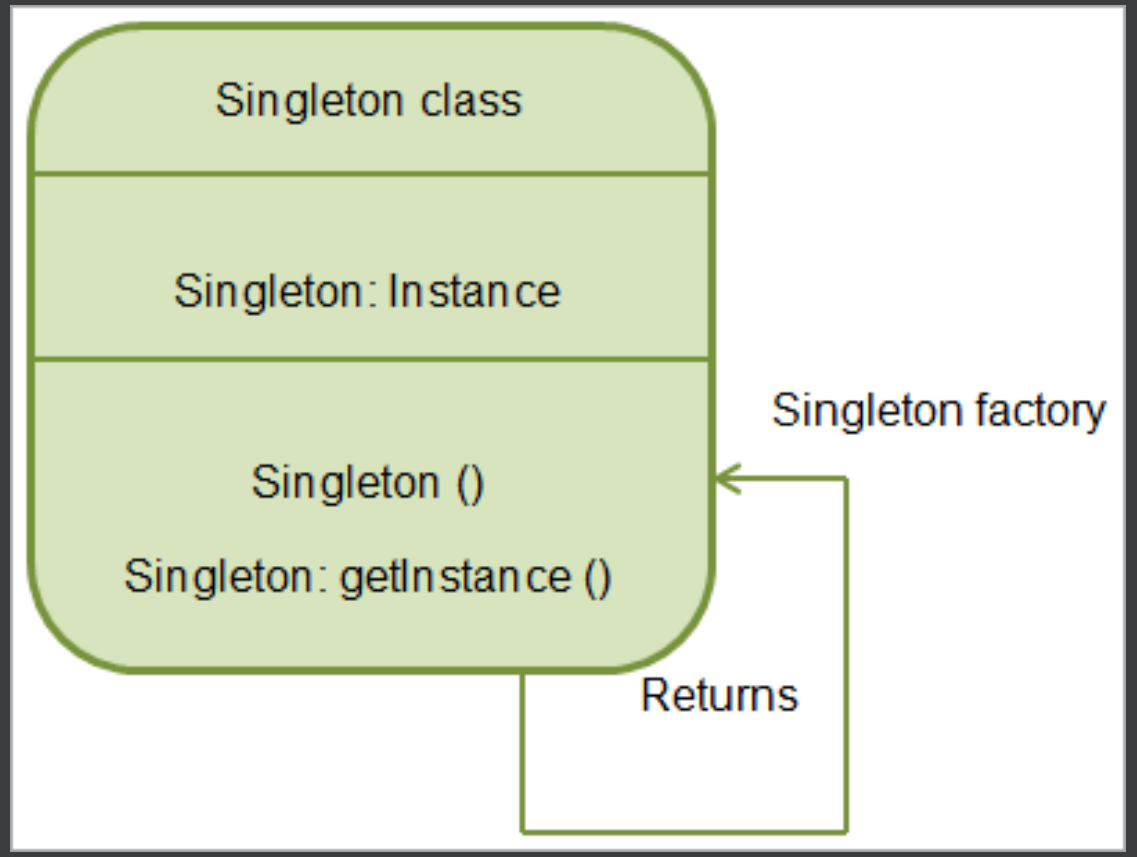
Structural Design Patterns mainly responsible for **assemble object and classes into a larger structure** making sure that these structure should be flexible

- Adapter
- Composite
- Bridge
- Decorator
- **Façade**
- **Flyweight**
- **Proxy**

Behavioral design Patterns

- **Chain of Responsibility**
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Singleton Pattern Class Diagram (Creational)



Singleton Pattern Example 1

```
class Database {
  constructor(data) {
    if (Database.exists) {
      return Database.instance;
    }
    this._data = data;
    Database.instance = this;
    Database.exists = true;
    return this;
  }

  getData() {
    return this._data;
  }

  setData(data) {
    this._data = data;
  }
}

// usage
const mongo = new Database("mongo");
console.log(mongo.getData()); // mongo

const mysql = new Database("mysql");
console.log(mysql.getData()); // mongo
```

Singleton Pattern Example 2

```
let _database = null;
class DataBase {
  constructor(typeOfConnection) {
    if (_database) {
      return _database;
    } else {
      this._typeOfConnection = typeOfConnection;
      _database = this;
    }
  }
  getDbInfo() {
    return _database._typeOfConnection;
  }
}

let db1 = new DataBase("mongo");
let db2 = new DataBase("Node");
console.log(db1._typeOfConnection);
console.log(db2._typeOfConnection);
```

Factory Method Pattern (Creational)

```
const createUser = (firstName, lastName) =>
({
  createdAt: Date.now(),
  firstName,
  lastName,
  fullName: `${firstName} ${lastName}`,
});

let a = createUser("Ahmed", "mohamed");
let a1 = createUser("Sarah", "Ahmed");
let a2 = createUser("Maged", "Samir");
let users = [a, a1, a2];
console.log(users);
```

Factory Pattern Example Using classes (Creational)

```
class factoryBook {
  constructor(Name, NumberOfP) {
    this._Name = Name;
    this._NumberOfP = NumberOfP;
  }
}
myBook = new factoryBook("asd", 12);
```

Factory Pattern Example Using classes

```
class BallFactory {
  constructor() {
    this.createBall = function(type) {
      let ball;
      if (type === 'football' || type === 'soccer') ball = new
      Football(); else if (type === 'basketball') ball = new
      Basketball();
      ball.roll = function() {
        return `The ${this._type} is rolling.`;
      };

      return ball;
    };
  }
}
```

Factory Pattern Example Using classes

```
class Football {
  constructor()  {
    this._type = 'football';
    this.kick = function() {
      return 'You kicked the football.';
    };
  }
}

class Basketball {
  constructor() {
    this._type = 'basketball';
    this.bounce = function() {
      return 'You bounced the basketball.';
    };
  }
}

// creating objects
const factory = new BallFactory();

const myFootball = factory.createBall('football');
const myBasketball = factory.createBall('basketball');

console.log(myFootball.roll()); // The football is rolling.
console.log(myBasketball.roll()); // The basketball is rolling.
console.log(myFootball.kick()); // You kicked the football.
console.log(myBasketball.bounce()); // You bounced the basketball.
```

Prototype Pattern syntax (Creational)

```
Object.create(proto);
Here we pass The object which should be the
prototype
of the newly-created object.

Object.create(proto, propertiesObject);
here we pass the new props
it's return A new object with the specified prototype
object and properties.
```

Prototype Pattern Example

```
// using Object.create as was recommended by ES standard
const car = {
  noOfWheels: 4,
  start() {
    return "started";
  },
  stop() {
    return "stopped";
  },
};

// Object.create(proto[, propertiesObject])

const myCar = Object.create(car, { owner: { value: "John" } });

console.log(myCar.__proto___=== car); // true
```

Flyweight Pattern Example (Structural)

```
// flyweight class
class Icecream {
  constructor(flavour, price) {
    this.flavour = flavour;
    this.price = price;
  }
}
```

Flyweight Pattern Example (Structural)

```
// factory for flyweight objects
class IcecreamFactory {
  constructor() {
    this._icecreams = [];
  }

  createIcecream(flavour, price) {
    let icecream =
    this.getIcecream(flavour); if (icecream)
    {
      return icecream;
    } else {
      const newIcecream = new Icecream(flavour, price);
      this._icecreams.push(newIcecream);
      return newIcecream;
    }
  }

  getIcecream(flavour) {
    return this._icecreams.find((icecream) => icecream.flavour ===
    flavour);
  }
}
```

Proxy Pattern Example (Structural)

```
// Proxy
// ES6 Proxy API = new Proxy(target, handler);
const cache = [];
const proxiedNetworkFetch = new Proxy(networkFetch, {
  apply(target, thisArg, args) {
    const urlParam = args[0];
    if (cache.includes(urlParam)) {
      return `${urlParam} - Response from cache`;
    } else {
      cache.push(urlParam);
          // send to function body
      return Reflect.apply(target, thisArg, args);
    }
  },
});
// usage
console.log(proxiedNetworkFetch("dogPic.jpg"));
// 'dogPic.jpg - Response from network'
console.log(proxiedNetworkFetch("dogPic.jpg"));
// 'dogPic.jpg - Response from cache'
```

https://javascript.info/proxy#proxy

Façade Pattern Example (Structural)

```
class FetchMusic {
  get resources() {
    return [
      { id: 1, title: "Music 1" },
      { id: 2, title: "Music 2" },
      { id: 3, title: "Music 3" },
    ];
  }

  fetch(id) {
    return this.resources.find((item) => item.id === id);
  }
}

class GetMovie {
  constructor(id) {
    return this.resources.find((item) => item.id === id);
  }

  get resources() {
    return [
      { id: 1, title: "frozen 1" },
      { id: 2, title: "frozen 2" },
      { id: 3, title: "frozen 3" },
    ];
  }
}
```

Façade Pattern Example (Structural)

```
const getTvShow = function (id) {
  const resources = [
    { id: 1, title: "TvShow 1" },
    { id: 2, title: "TvShow 2" },
    { id: 3, title: "TvShow 3" },
  ];

  return resources.find((item) => item.id === id);
};

const booksResource = [
  { id: 1, title: "JS" },
  { id: 2, title: "HTML" },
  { id: 3, title: "CSS" },
];
```

Façade Pattern Example (using pattern) (Structural)

```
class CultureFacade {
  _findMusic(id) {
    const db = new FetchMusic();
    return db.fetch(id);
  }

  _findMovie(id) {
    return new GetMovie(id);
  }

  _findTVShow(id) {
    return getTvShow(id);
  }

  _findBook(id) {
    return booksResource.find((item) =>
item.id === id);
  }
}
```

Chain of Responsibility Pattern (Behavioural)

```
x = "Hello ya Cairo".split(" ").reverse().join("_");
console.log(x);
```

Chain of Responsibility Pattern

```
class CumulativeSum {
  constructor(intialValue = 0) {
    this.sum = intialValue;
  }

  add(value) {
    this.sum += value;
    return this;
  }
}

// usage
const sum1 = new CumulativeSum();
console.log(sum1.add(10).add(2).add(50).sum); // 62

const sum2 = new CumulativeSum(10);
console.log(sum2.add(10).add(20).add(5).sum); // 45
```

Chain of Responsibility Pattern

```
class CumulativeSum {
  constructor(intialValue = "") {
    this.txt = intialValue;
  }
  checkval(value) {
    if (value.includes("a")) {
      this.txt = value;
      return this;
    } else {
      console.error("the first letter is not a ");
    }
  }
  addnumber(num) {
    if (this.txt.includes("s")) {
      this.txt += num;
      return this;
    } else {
      // console.error("no s includes ");
      throw new Error("Hello im Error");
    }
  }
  print() {
    console.log(this.txt);
    console.log(this.txt);
    console.log(this.txt);
    console.log(this.txt);
  }
} // usage
const sum1 = new CumulativeSum();
console.log(sum1.checkval("aad").addnumber(1).print());
```

Task 1

Turn this class into a singleton, to ensure that only one DBConnection instance can exist.

```
class DBConnection {
  constructor(uri) {
    this.uri = uri;
  }

  connect() {
    console.log(`DB ${this.uri} has been connected!`);
  }

  disconnect() {
    console.log("DB disconnected");
  }
}

const connection = new DBConnection("mongodb://...");
```

Task 2

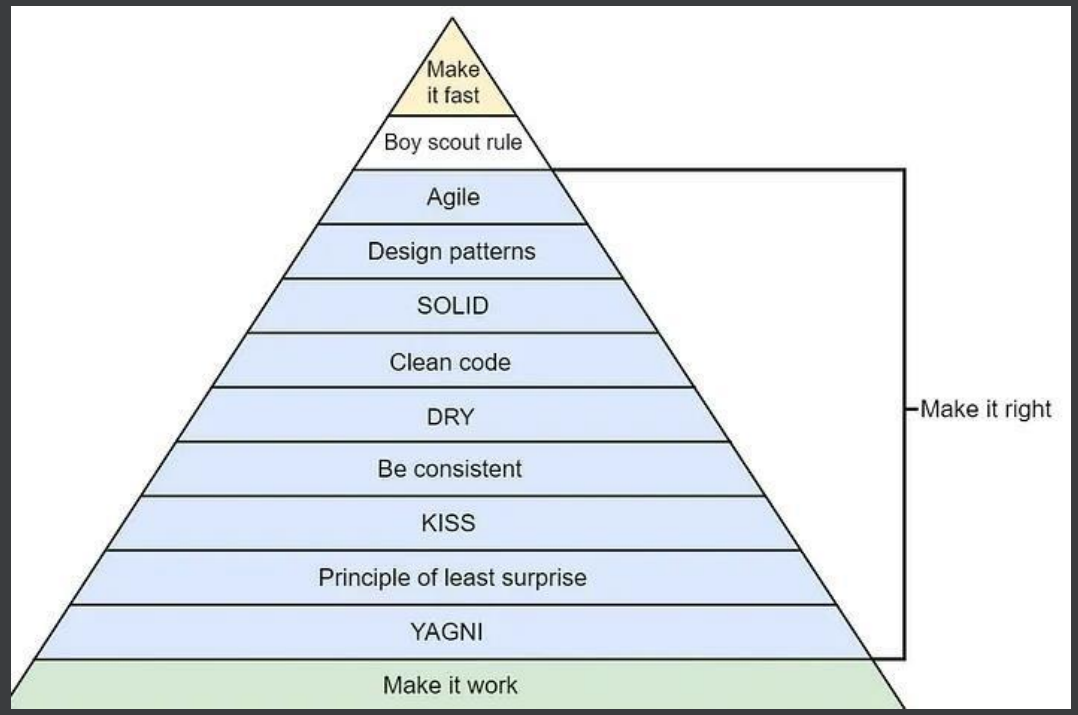
```
const book1 = {
  title: "Harry Potter",
  author: "JK Rowling",
  isbn: "AB123",
};

const book2 = {
  title: "The Great Gatsby",
  author: "F. Scott Fitzgerald",
  isbn: "CD456",
};

const book3 = {
  title: "Moby-Dick",
  author: "Herman Melville",
  isbn: "EF789",
};

const book4 = {
  title: "Harry Potter",
  author: "JK Rowling",
  isbn: "AB123",
};

const book5 = {
  title: "The Great Gatsby",
  author: "F. Scott Fitzgerald",
  isbn: "CD456",
};

books = [book1, book2, book3, book4, book5];
console.log(books);
```

A Deep Dive into Shallow Copy and Deep Copy in JavaScript
https://javascript.plainenglish.io/shallow-copy-and-deep-copy-in-javascript-a0a04104ab5c

Proxy and Reflect
https://javascript.info/proxy#proxy

Software Development Principles
https://www.boldare.com/blog/kiss-yagni-dry-principles/#what-is-the-kiss-principle?
https://thefullstack.xyz/dry-yagni-kiss-tdd-soc-bdfu
https://medium.com/@bartoszkrajka/principle-of-software-development-principles-f0143d6f405

Performance, rendering and react patterns
https://reactpatterns.com/



Education Resources Sheet

https://drum.io/milicodes?fbclid=PAAaZttHMfo6uINqqZL0Mm995Y9C7qU4dDkV-ITGTIjRNfU1dBpqS1pwpkf_4
https://devdojo.com/arpit/github-repos-to-become-better-javascript-developer
https://github.com/lydiahallie/javascript-questions
https://www.amazon.com/JavaScript-Beginner-Professional-building-interactive/dp/1800562527
https://www.kickresume.com/en/?tap_a=74981-376a6f&tap_s=2688785-c4334f
https://equable-methane-c17.notion.site/7093e41669aa433590150a8c76bcd3d4?v=8822d291a53b49d38611a4b64f42940b
https://www.w3resource.com/javascript-exercises/javascript-array-exercises.php
https://github.com/milicodes?tab=repositories
https://www.milipernia.com/
https://learnjavascript.online/knowledge-map.html?ref=producthunt
https://exploringjs.com/
https://estelle.github.io/
https://developer.mozilla.org/en-US/
https://snipcart.com/blog/javascript-practice-exercises
https://www.w3resource.com/javascript-exercises/
cvflow

Remote Work

https://angel.co/login
https://www.workingnomads.com/remote-development-jobs
https://www.numbeo.com/common/api.jsp
https://remotive.com/
https://weworkremotely.com/
https://himalayas.app/jobs/developer
https://remoteok.com/remote-javascript+junior-jobs
https://remote.co/remote-jobs/search/?search_keywords=javascript
https://jobs.getmimo.com/
https://djinni.co/login
https://arc.dev/
https://anywhere.epam.com/en
https://remoteplatz.com/
https://remoteok.com/
https://justremote.co/
https://angellisttalent.statuspage.io/
https://weworkremotely.com/
https://hired.com/
https://www.flexjobs.com/jobs/featured
https://www.upwork.com/
https://jobspresso.co/
https://dailyremote.com/
https://remoteleaf.com/