

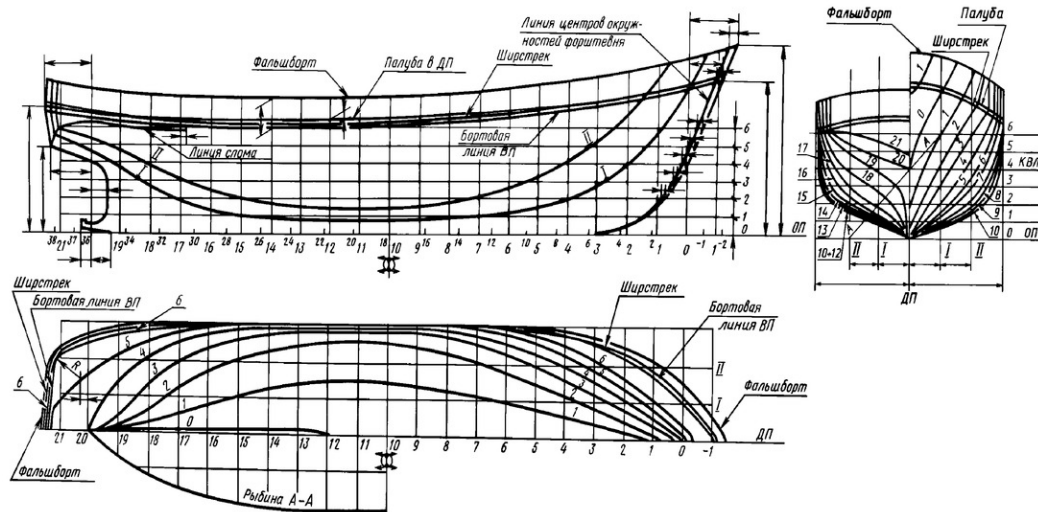


Паттерны проектирования JavaScript

17.08.2020

ЧТО ТАКОЕ ПАТТЕРН?
ЧЕМ ПАТТЕРН ОТЛИЧАЕТСЯ ОТ АЛГОРИТМА?

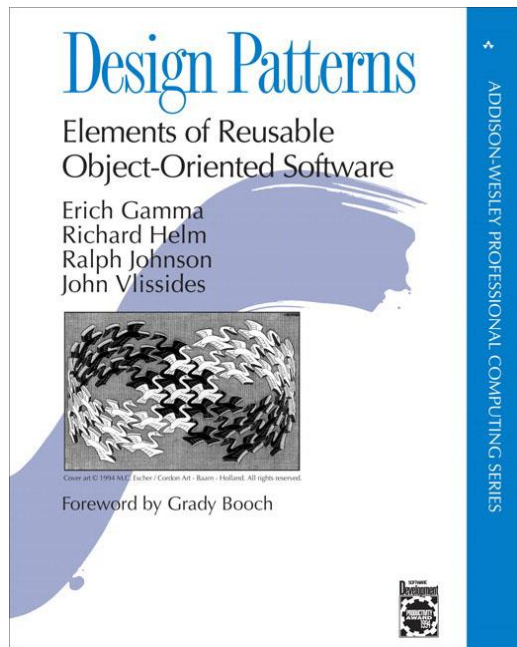
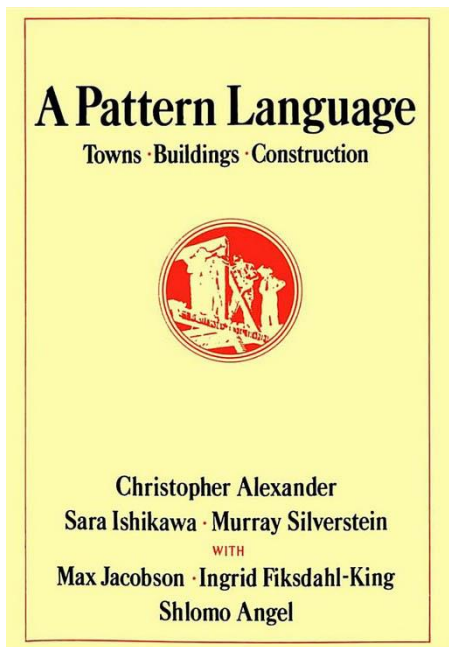
Что ты такое



Алгоритм — последовательность действий для решения конкретной задачи

Паттерн — типичное решение для часто возникающей проблемы

Как всё начиналось



1977 год — впервые описана концепция паттернов

1994 год — книга "Банды Четырёх": паттерны, решающие проблемы ОО-дизайна

Какие бывают паттерны

1 ПОРОЖДАЮЩИЕ

- Создание новых объектов

2 СТРУКТУРНЫЕ

- Взаимосвязи между объектами

3 ПОВЕДЕНЧЕСКИЕ

- Взаимодействие между объектами

Знание паттернов позволяет нам:



Экономить время, не изобретая заново велосипед



Делать меньше просчетов при проектировании



Общаться с другими разработчиками "на одном языке"

Паттерны - это прекрасно, но:



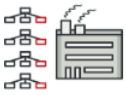
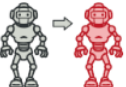


Не используйте паттерны там, где вы можете без них обойтись

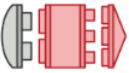


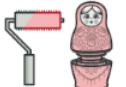

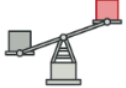


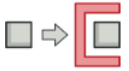
Не используйте паттерны бездумно, не адаптируя их под ваши задачи





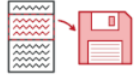

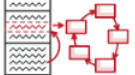
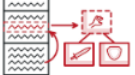




Паттерны — это костыль при недостаточном уровне абстракции

	
Фабричный метод Factory Method	Абстрактная фабрика Abstract Factory
	
Строитель Builder	Прототип Prototype
	
Одиночка Singleton	Модуль Module

	
Адаптер Adapter	Мост Bridge
	
Компоновщик Composite	Декоратор Decorator
	
Фасад Facade	Легковес Flyweight


Заместитель Proxy

			
Цепочка обязанностей Chain of Responsibility	Команда Command	Итератор Iterator	Посредник Mediator
			
Снимок Memento	Наблюдатель Observer	Состояние State	Стратегия Strategy
			
Шаблонный метод Template Method	Посетитель Visitor		

ПОРОЖДАЮЩИЕ
СТРУКТУРНЫЕ
ПОВЕДЕНЧЕСКИЕ

МОДУЛЬ


Частота использования: 

Сложность написания: 

Что делает: **Модуль** - порождающий паттерн, позволяющий скрывать приватную информацию

Зачем использовать: В JavaScript нет встроенных средств, позволяющих скрывать методы и свойства классов

МОДУЛЬ



```
var Module = (function () {  
    var _privateMethod = function () {  
  
    };  
    var publicMethod = function () {  
  
    };  
    return {  
        publicMethod: publicMethod,  
    }  
})();
```

ПРОТОТИП

Частота использования: 

Сложность написания: 

Что делает: **Прототип** - порождающий паттерн, позволяющий копировать объекты, не вдаваясь в подробности их реализации

Зачем использовать: В JavaScript нет встроенных средств, позволяющих скрывать методы и свойства классов



```
class Robot {  
  constructor(color, capacity) {  
    this.color = color;  
    this.capacity = capacity;  
  }  
}  
  
let r100 = new Robot('red', 100);  
let r200 = new Robot('white', 1000);
```

ФАБРИКА

Частота использования: 

Сложность написания: 

Что делает: **Фабрика** - порождающий паттерн, не требующий явного использования конструктора

Зачем использовать: Для создания сложных объектов, зависящих от динамических факторов

ФАБРИКА

```
class Bmw {
  constructor(model, price, maxSpeed) {
    this.model = model;
    this.price = price;
    this.maxSpeed = maxSpeed;
  }
}

class BmwFactory {
  create(type) {
    if (type === 'X5') return new Bmw(type, 108000, 300);
    if (type === 'X6') return new Bmw(type, 111000, 320);
  }
}

const factory = new BmwFactory();

const x5 = factory.create('X5');
const x6 = factory.create('X6');

console.log(x5);
console.log(x6);
```

ФАБРИЧНЫЙ МЕТОД

Частота использования: 

Сложность написания: 

Что делает: **Фабричный метод** - порождающий паттерн, определяющий общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов

Зачем использовать: Когда заранее неизвестны типы и зависимости объектов в нашем коде

ФАБРИЧНЫЙ МЕТОД

```
function Car(options) {
  this.doors = options.doors || 4;
  this.state = options.state || 'brand new';
  this.color = options.color || 'silver';
}

function Truck(options) {
  this.state = options.state || 'used';
  this.wheelSize = options.wheelSize || 'large';
  this.color = options.color || 'blue';
}
```

```
function VehicleFactory() {}
VehicleFactory.prototype.vehicleClass = Car;
VehicleFactory.prototype.createVehicle = function (options) {
  switch (options.vehicleType) {
    case 'car':
      this.vehicleClass = Car;
      break;
    case 'truck':
      this.vehicleClass = Truck;
      break;
  }

  return new this.vehicleClass(options);
};
```

ФАБРИЧНЫЙ МЕТОД



```
var carFactory = new VehicleFactory();  
var car = carFactory.createVehicle({  
  vehicleType: 'car',  
  color: 'yellow',  
  doors: 6,  
});
```

```
console.log(car instanceof Car);  
console.log(car);
```



```
var movingTruck = carFactory.createVehicle({  
  vehicleType: 'truck',  
  state: 'like new',  
  color: 'red',  
  wheelSize: 'small',  
});
```

```
console.log(movingTruck instanceof Truck);  
console.log(movingTruck);
```

ФАБРИЧНЫЙ МЕТОД



```
function TruckFactory() {}  
TruckFactory.prototype = new VehicleFactory();  
TruckFactory.prototype.vehicleClass = Truck;  
  
var truckFactory = new TruckFactory();  
var myBigTruck = truckFactory.createVehicle({  
  state: 'omg..so bad.',  
  color: 'pink',  
  wheelSize: 'so big',  
});  
  
console.log(myBigTruck instanceof Truck);  
console.log(myBigTruck);
```

Что почитать по теме:



["Learning JavaScript Design Patterns" by Addy Osmani](#)



["Паттерны для масштабируемых JavaScript-приложений", Эдди Османи](#)



["Паттерны проектирования", Эрик Фримен, Элизабет Робсон](#)