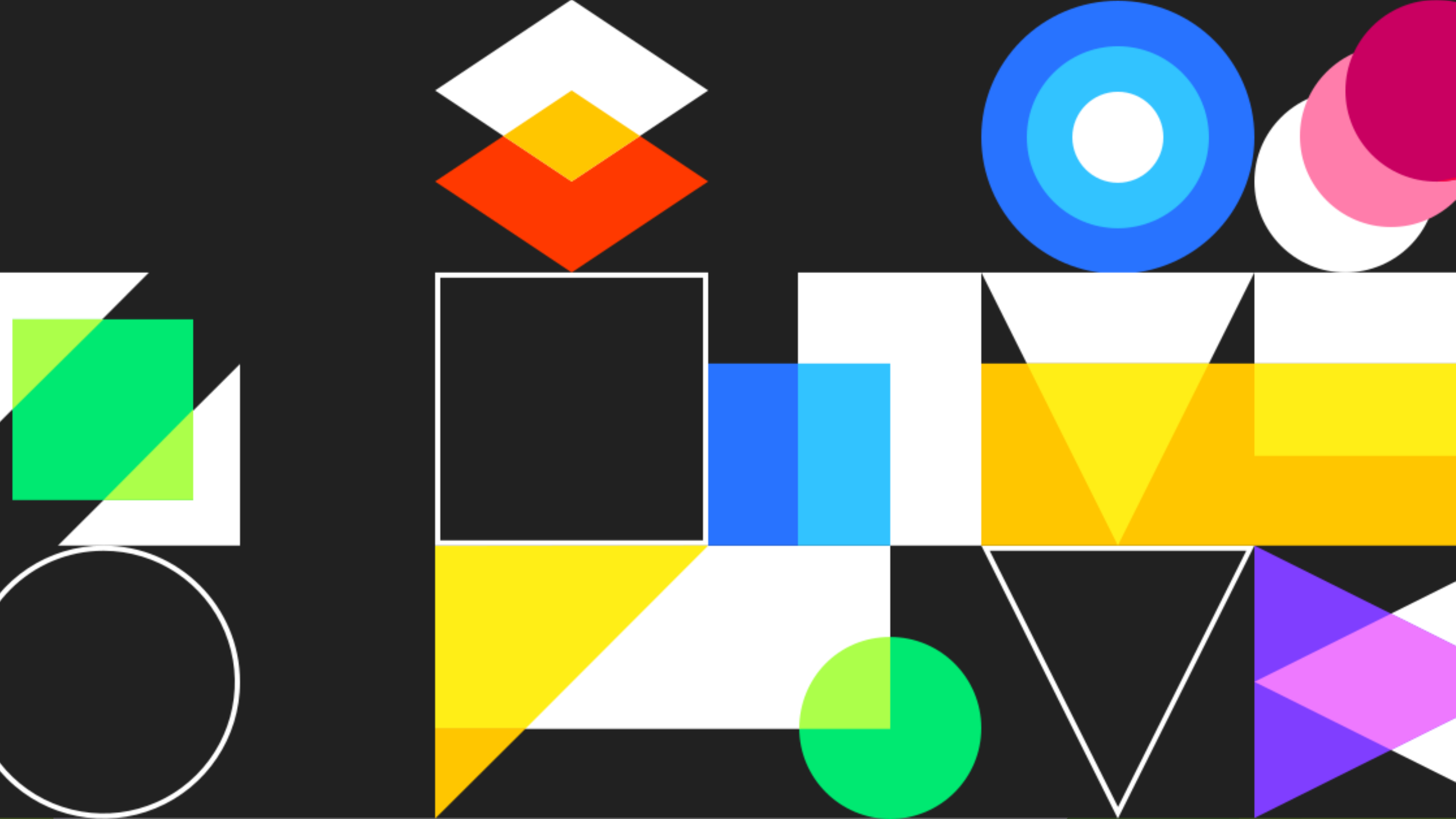
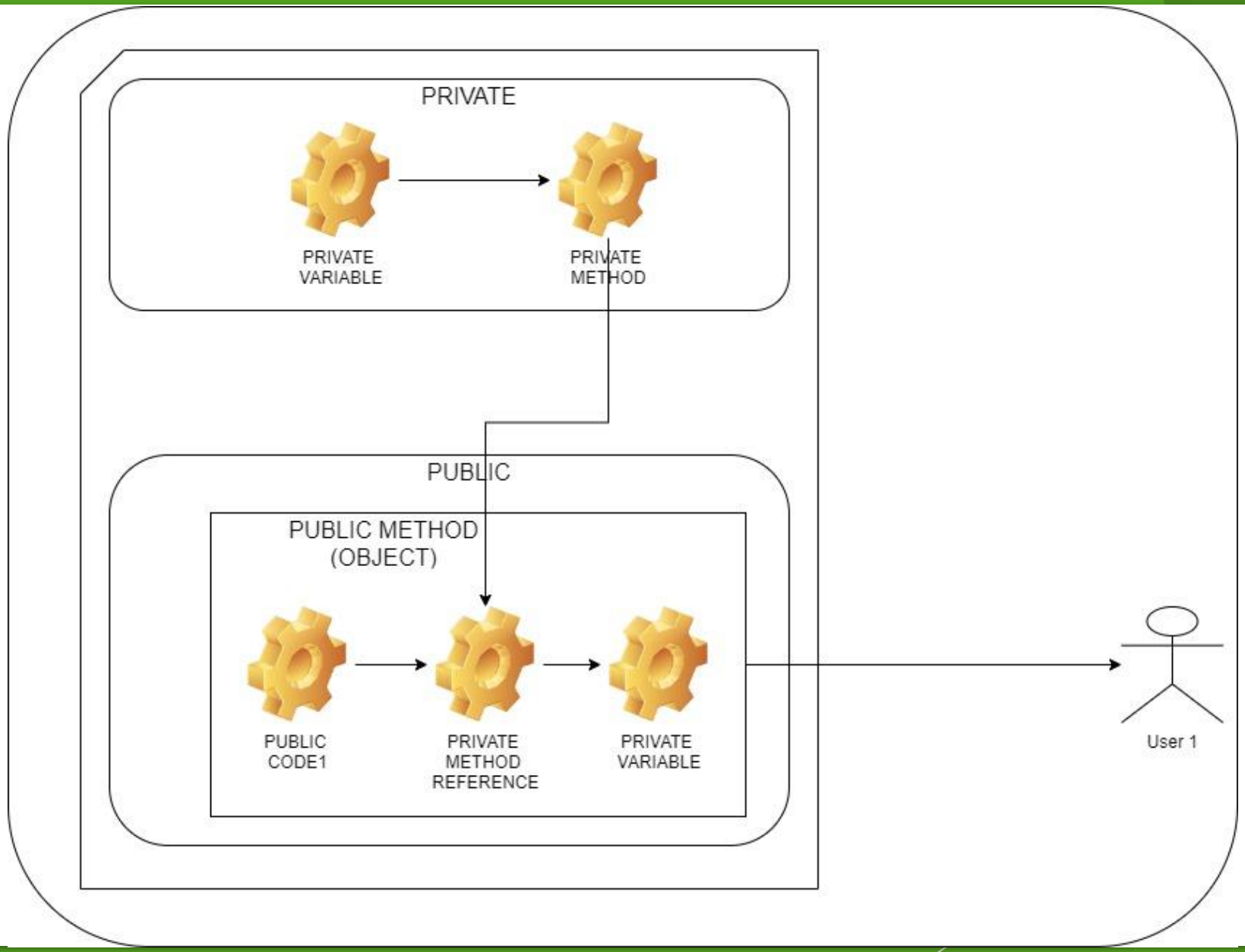
S2 Z3

Design Patterns



Design Patterns

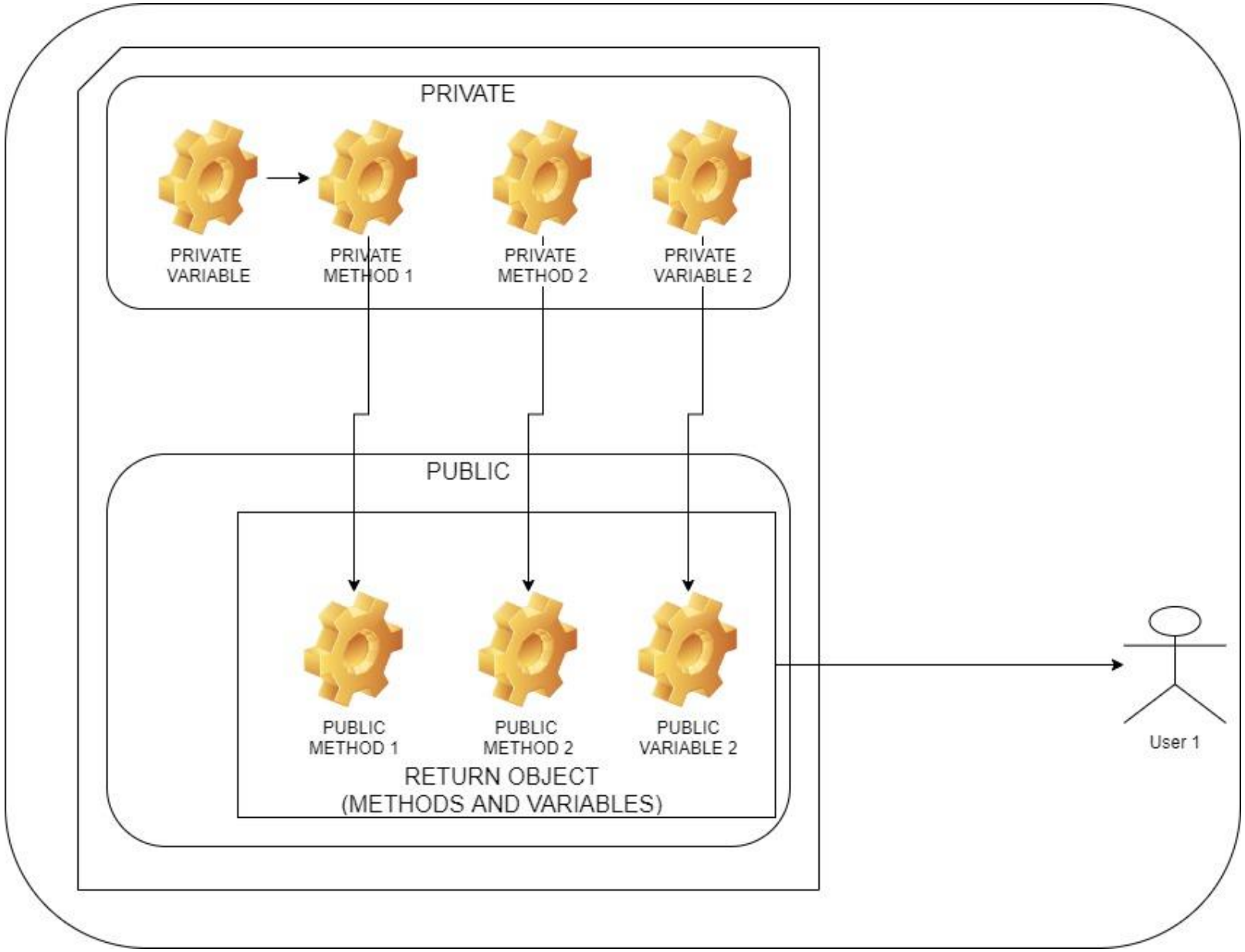
- Modules x3
- Singleton x5
- Factory x3
- Decorator x4


PRIVATE
PRIVATE VARIABLE
PRIVATE METHOD
PUBLIC
PUBLIC METHOD (OBJECT)
PUBLIC CODE1
PRIVATE METHOD REFERENCE
PRIVATE VARIABLE
User 1

Module / Closure

```
const myModule = (function() {
    const privateVariable = "Hello World";

    function privateMethod() {
        console.log(privateVariable);
    }

    return {
        publicMethod: function() {
            console.log("publicMethod start");
            privateMethod();
            console.log("publicMethod end");
        }
    };
})();

console.log(myModule);

myModule.publicMethod();
```

```
publicMethod start
Hello World
publicMethod end
```


PRIVATE
PRIVATE VARIABLE
PRIVATE METHOD 1
PRIVATE METHOD 2
PRIVATE VARIABLE 2
PUBLIC
PUBLIC METHOD 1
PUBLIC METHOD 2
PUBLIC VARIABLE 2
RETURN OBJECT
(METHODS AND VARIABLES)
User 1

Revealing Module ½ (definition)

```
const myRevealingModule = (function() {
    let privateVar = "Peter";
    const publicVar = "Hello World";

    function privateFunction() {
        console.log("Name: " + privateVar);
    }

    function publicSetName(name) {
        privateVar = name;
    }

    function publicGetName() {
        privateFunction();
    }

    return {
        setName: publicSetName,
        greeting: publicVar,
        getName: publicGetName
    };
})();
```

Revealing Module 2/2 (usage)

```
console.log(myRevealingModule);

myRevealingModule.setName("Mark");
myRevealingModule.getName();
```

```
{setName: ƒ, greeting: 'Hello World', getName: ƒ}
```

Name: Mark

ES6 Module 1/3 (definition)

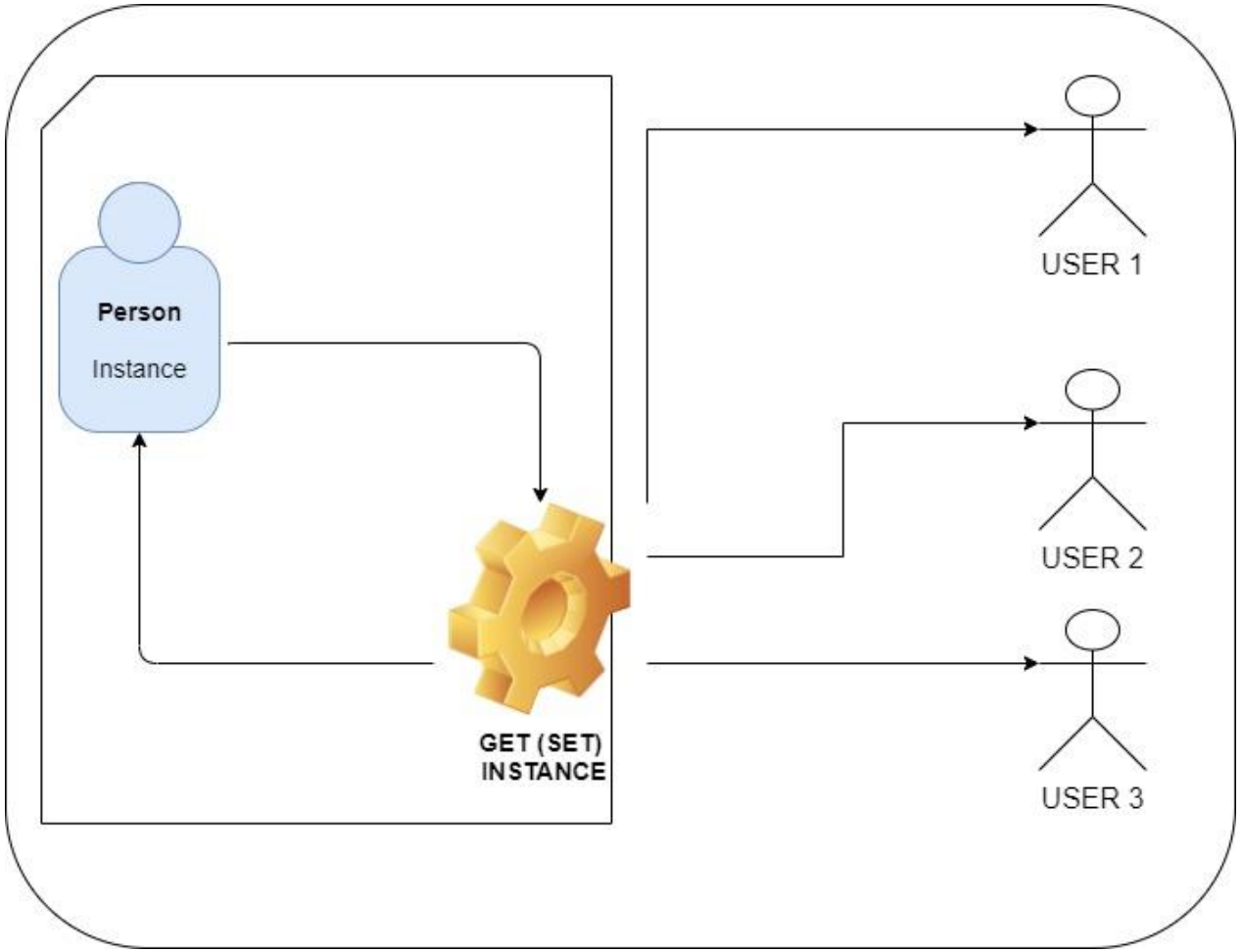
```
const greeting = "Hello World";
let callCounter = 0;

function sayHelloFirstTime() {
    if (callCounter++ === 0) {
        console.log(greeting);
    }
}

function privateLog() {
    console.log("Private Function");
}

function multiply(num1, num2) {
    sayHelloFirstTime();
    console.log("Multiply:", num1, num2);
    return num1 * num2;
}
```

ES6 Module 2/3 (definition)

```
function subtract(num1, num2) {
    sayHelloFirstTime();
    console.log("Subtract:", num1, num2);
    return num1 - num2;
}

module.exports = {
    sum: sum,
    subtract: subtract,
    multiply: multiply,
    divide: divide
};
```

ES6 Module 3/3 (usage)

```
let mathUtils = require("./utils/utils");

console.log(mathUtils);

console.log(mathUtils.sum(3, 7));
console.log(mathUtils.subtract(3, 7));
console.log(mathUtils.multiply(3, 7));
console.log(mathUtils.divide(3, 7));
```


Person
Instance
GET (SET)
INSTANCE
USER 1
USER 2
USER 3

Singleton

```
function User() {
    this.name = "Peter";
    this.age = 25;
}

const user1 = new User();
const user2 = new User();
console.log(user1 === user2);
```

false

Singleton

```
let instance = null;

function User() {
    if (instance) {
        return instance;
    }

    instance = this;
    this.name = "Peter";
    this.age = 25;

    return instance;
}

const user1 = new User();
const user2 = new User();
console.log(user1 === user2);
```

true

Singleton

```
const singleton = (function() {
    let instance;

    function init() {
        return {
            name: "Peter",
            age: 24
        };
    }
    return {
        getInstance: function() {
            if (!instance) {
                instance = init();
            }

            return instance;
        }
    };
})();

const instanceA = singleton.getInstance();
const instanceB = singleton.getInstance();
console.log(instanceA === instanceB);
```

true

Singleton (1/2)

```
class Person {
    constructor() {
        this.name = "Peter";
        this.age = 25;
    }
}

const singleton = (function() {
    let instance;
    function init() {
        return new Person();
    }

    return {
    getInstance: function() {
        if (!instance) {
            instance = init();
        }

        return instance;
        }
    };
})();
```

Singleton (2/2)

```
const instanceA = singleton.getInstance();
const instanceB = singleton.getInstance();

console.log(instanceA === instanceB);
console.log(instanceA === instanceC);
```

true

false

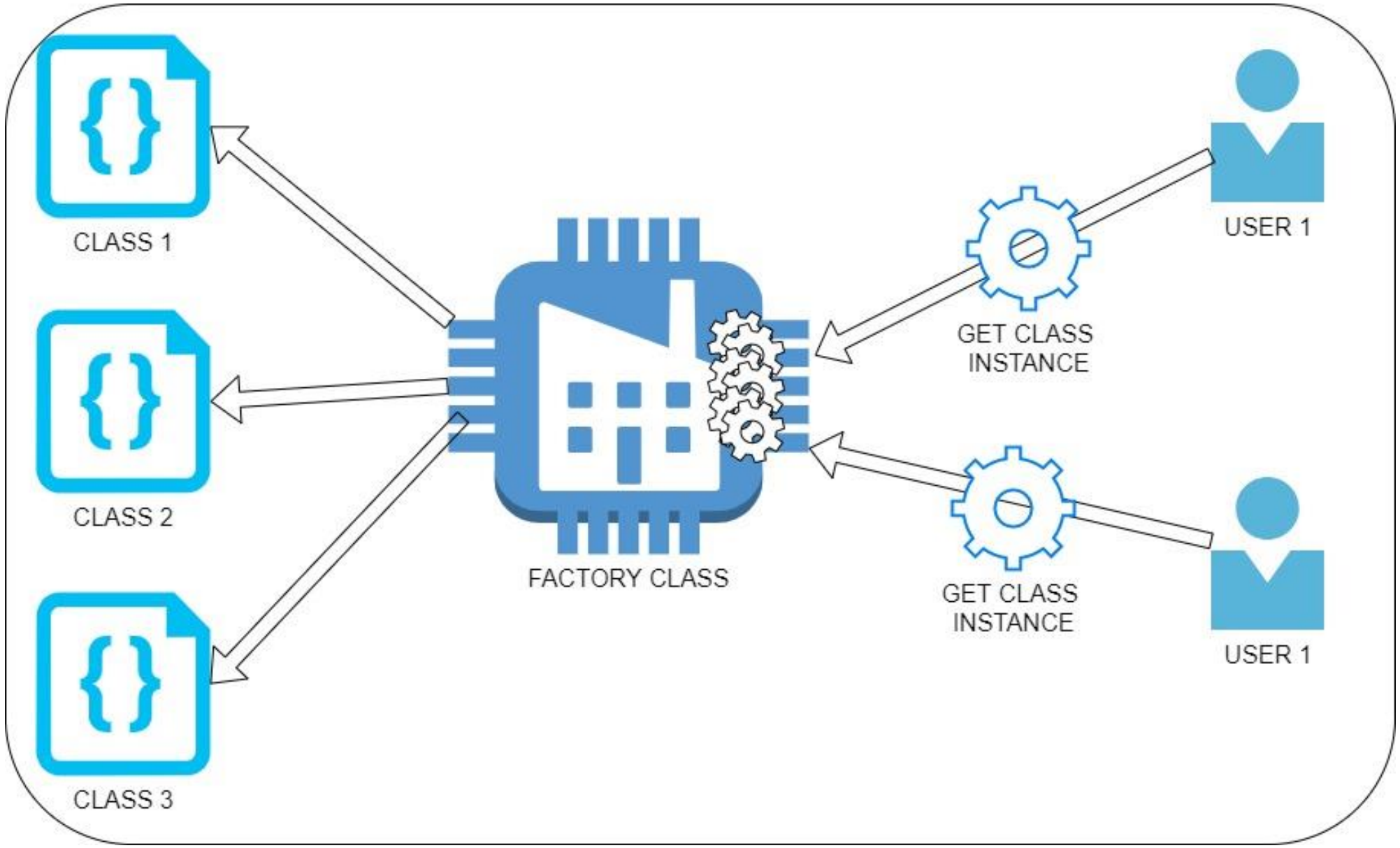
Singleton (1/2)

```
const singleton = (function() {
    let instance;
    function init() {
        return new Person();
    }

    class Person {
        constructor() {
            this.name = "Peter";
            this.age = 25;
        }
    }

    return {
        getInstance: function() {
        if (!instance) {
            instance = init();
        }

        return instance;
        }
    };
})();
```

Singleton (2/2)

```
const instanceA = singleton.getInstance();
const instanceB = singleton.getInstance();

console.log(instanceA === instanceB);
```

true


CLASS 1
CLASS 2
CLASS 3
FACTORY CLASS
GET CLASS INSTANCE
GET CLASS INSTANCE
USER 1
USER 1

Factory Basic ½ (definition)

```
class Car {
    constructor(options) {
        this.doors = options.doors || 4;
        this.state = options.state || "brand new";
        this.color = options.color || "white";
    }
}
class Truck {
    constructor(options) {
        this.doors = options.doors || 4;
        this.state = options.state || "used";
        this.color = options.color || "black";
    }
}
class VehicleFactory {
    createVehicle(options) {
        if (options.vehicleType === "car") {
            return new Car(options);
        } else if (options.vehicleType === "truck") {
            return new Truck(options);
        }
    }
}
```

Factory Basic 2/2 (usage)

```
const factory = new VehicleFactory();

const car = factory.createVehicle({
    vehicleType: "car",
    doors: 4,
    color: "silver",
    state: "Brand New"
});

const truck = factory.createVehicle({
    vehicleType: "truck",
    doors: 2,
    color: "white",
    state: "used"
});

console.log(car);

console.log(truck);
```

Car {doors: 4, state: "Brand New", color: "silver"}
Truck {doors: 2, state: "used", color: "white"}

Factory Advanced 1/3 (objects definition)

```
class Vehicle {
    constructor(vehicleType) {
        this.vehicleType = vehicleType;
    }
}

class Car extends Vehicle {
    constructor(options) {
        super(options.vehicleType);
        this.doors = options.doors || 4;
        this.state = options.state || "brand new";
        this.color = options.color || "white";
    }
}

class Truck extends Vehicle {
    constructor(options) {
        super(options.vehicleType);
        this.doors = options.doors || 4;
        this.state = options.state || "used";
        this.color = options.color || "black";
    }
}
```

Factory Advanced 2/3 (factory definition)

```
class VehicleFactory {
   createVehicle(options) {
      if (options.vehicleType === "car") {
         return new Car(options);
      } else if (options.vehicleType === "truck")
      {
         return new Truck(options);
      }
   }
}
```

Factory Advanced 3/3 (usage)

```
const factory = new VehicleFactory();

const car = factory.createVehicle({
    vehicleType: "car",
    doors: 4,
    color: "silver",
    state: "Brand New"
});

const truck = factory.createVehicle({
    vehicleType: "truck",
    doors: 2,
    color: "white",
    state: "used"
});

console.log(car);

console.log(truck);
```

Car {vehicleType: "car", doors: 4, state: "Brand New", color: "silver"}
Truck {vehicleType: "truck", doors: 2, state: "used", color: "white"}

Factory ½ Definition

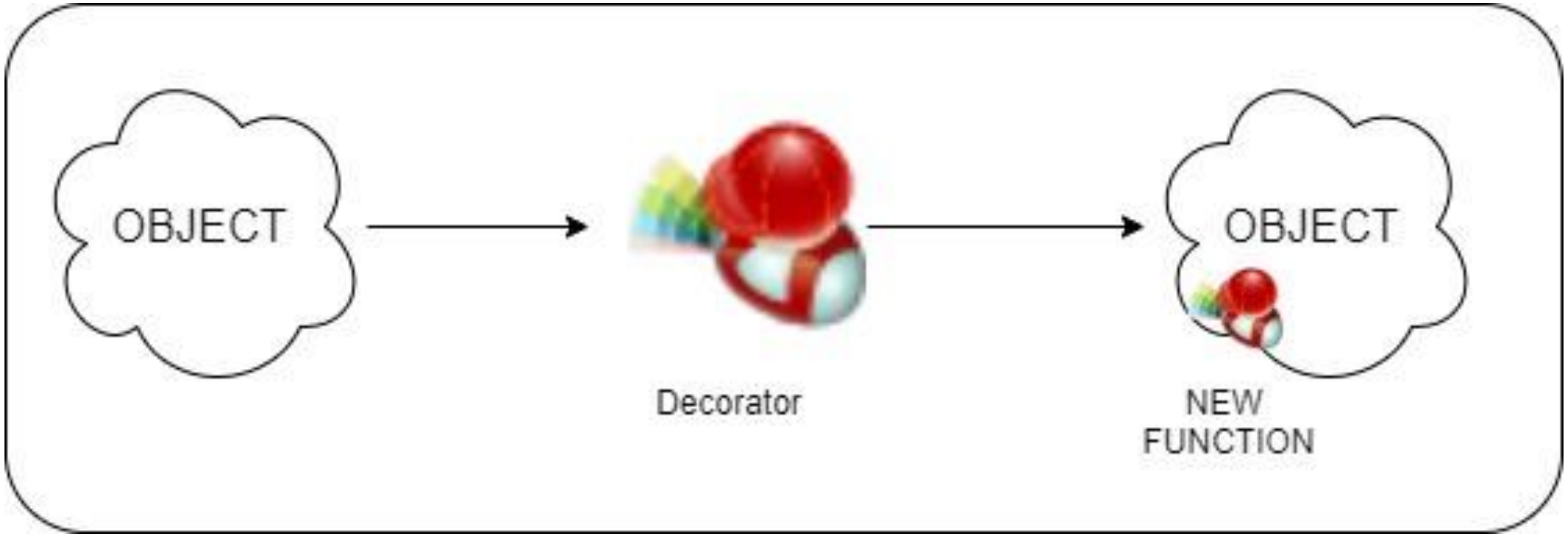
```
const factory = (function() {
    class Car {
    constructor(options) {
        this.doors = options.doors || 4;
        this.state = options.state || "brand new";
        this.color = options.color || "white";
        }
    }
    class Truck {
    constructor(options) {
        this.doors = options.doors || 4;
        this.state = options.state || "used";
        this.color = options.color || "black";
        }
    }

    return {
        createVehicle: options => {
            if (options.vehicleType === "car") {
                return new Car(options);
            } else if (options.vehicleType === "truck") {
                return new Truck(options);
            }
        }
    };
})();
```

Factory ½ Definition

```
const car = factory.createVehicle({
    vehicleType: "car",
    doors: 4,
    color: "silver",
    state: "Brand New"
});
const truck = factory.createVehicle({
    vehicleType: "truck",
    doors: 2,
    color: "white",
    state: "used"
});

console.log(car);

console.log(truck);

console.log(
// class accesable. Good idea?
new Car({
    doors: "Yes",
    color: 666,
    state: Date.now()
    })
);
```

Car {doors: 4, state: "Brand New", color: "silver"}
Truck {doors: 2, state: "used", color: "white"}

ReferenceError: Car is not defined


OBJECT
Decorator
OBJECT
NEW
FUNCTION

Decorator 1/3

```
class Car {
   constructor() {
      this.cost = function() {
         return 20000;
      };

      this.desc = "basic";
   }
}
```

Decorator 2/3

```
function carWithAC(car) {
    car.hasAC = true;
    const prevCost = car.cost();
    car.cost = function() {
        return prevCost + 500;
    };

    car.desc += " AC";
}

function carWithAutoTransmission(car) {
    car.hasAutoTransmission = true;
    const prevCost = car.cost();
    car.cost = function() {
        return prevCost + 2000;
    };

    car.desc += " AutoT";
}
```

Decorator 3/3

```
const car = new Car();
console.log(car.cost());
carWithAC(car);
carWithAutoTransmission(car);

console.log(car.cost());
console.log(car.desc);
console.log(car);
```

```
20000
22500
basic AC AutoT
Car {cost: , desc: "basic AC AutoT", hasAC: true, hasAutoTransmission: true}
```

Decorator

```
class Car {
    constructor() {
        this.cost = function() {
            return 20000;
        };

        this.desc = "basic";
    }
}

const car = new Car();
car.autoPark = () => console.log("Auto parking...");

console.log(car);
car.autoPark();
```

Car {cost: , desc: "basic", autoPark: }
Auto parking...

Decorator

```
class Car {
    constructor() {
        this.cost = function() {
            return 20000;
        };

        this.desc = "basic";
    }
}

function addParking(car) {
    car.autoPark = () => console.log("Auto parking...");
}

const car = new Car();

addParking(car);
console.log(car);
car.autoPark();
```

```
Car {cost: , desc: "basic", autoPark: }
Auto parking...
```

Decorator

```
class Car {
    constructor() {
        this.cost = function() {
            return 20000;
    };

        this.desc = "basic";
    }
}

function addParking() {
    Car.prototype.autoPark = () => console.log("Auto parking...");
}

const car = new Car();
addParking();

console.log(car);
car.autoPark();
```

Car {cost: , desc: "basic", autoPark: }
Auto parking...