SOLID Principle

1. Single Responsibility Principle (SRP)

Tujuan: Setiap kelas atau modul harus memiliki satu tanggung jawab yang jelas dan terdefinisi dengan baik.

Contoh:

```
// Sebelum menerapkan SRP
class UserSettings {
  constructor(user) {
    this.user = user
  }

  changeUsername(username) {
    this.user.username = username
  }

  changeEmail(email) {
    this.user.email = email
  }

  saveUser() {
    database.save(this.user)
  }
}

// Setelah menerapkan SRP
class User {
  constructor(username, email) {
    this.username = username
    this.email = email
  }
}

class UserDB {
  static save(user) {
    database.save(user)
  }
}
```

2. Open/Closed Principle (OCP)

Tujuan: Entitas perangkat lunak harus terbuka untuk ekstensi, tetapi tertutup untuk modifikasi.

Contoh:

```
// Sebelum menerapkan OCP
class Rectangle {
  constructor(width, height) {
    this.width = width
    this.height = height
  }
}

function calculateArea(rectangles) {
  return rectangles.reduce((area, rectangle) => {
    return area + rectangle.width * rectangle.height
  }, 0)
}

// Setelah menerapkan OCP
class Shape {
  area() {}
}

class Rectangle extends Shape {
  constructor(width, height) {
    super()
    this.width = width
    this.height = height
  }

  area() {
    return this.width * this.height
  }
}

function calculateTotalArea(shapes) {
  return shapes.reduce((area, shape) => {
    return area + shape.area()
  }, 0)
}
```

3. Liskov Substitution Principle (LSP)

Tujuan: Subtipe harus dapat menggantikan tipe dasarnya tanpa mengubah kebenaran program.

Contoh:

```
class Bird {
  fly() {
    console.log('I can fly!')
  }
}

class Duck extends Bird {}

class Penguin extends Bird {
  fly() {
    throw new Error('Cannot fly!')
  }
}

function makeBirdFly(bird) {
  bird.fly()
}

// Setelah menerapkan LSP
class FlyingBird extends Bird {
  fly() {
    console.log('I can fly!')
  }
}

class NonFlyingBird extends Bird {}

class Duck extends FlyingBird {}

class Penguin extends NonFlyingBird {}
```

4. Interface Segregation Principle (ISP)

Tujuan: Klien tidak seharusnya dipaksa untuk bergantung pada antarmuka yang tidak mereka gunakan.

Contoh:

```
// Sebelum menerapkan ISP
class Worker {
  work() {
    // ...working
  }

  eat() {
    // ...eating in lunch break
  }
}

// Setelah menerapkan ISP
class Workable {
  work() {}
}

class Eatable {
  eat() {}
}

class Worker extends Workable {}

class Robot extends Workable {}
```

5. Dependency Inversion Principle (DIP)

Tujuan: Bergantung pada abstraksi, bukan konkrit.

Contoh:

```
// Sebelum menerapkan DIP
class LightBulb {
  turnOn() {
    console.log('LightBulb: turned on...')
  }

  turnOff() {
    console.log('LightBulb: turned off...')
  }
}

class ElectricPowerSwitch {
  constructor(bulb) {
    this.bulb = bulb
    this.isOn = false
  }

  press() {
    if (this.isOn) {
      this.bulb.turnOff()
      this.isOn = false
    } else {
      this.bulb.turnOn()
      this.isOn = true
    }
  }
}
```

```
// Setelah menerapkan DIP
class SwitchableDevice {
  turnOn() {}
  turnOff() {}
}

class LightBulb extends SwitchableDevice {
  turnOn() {
    console.log('LightBulb: turned on...')
  }

  turnOff() {
    console.log('LightBulb: turned off...')
  }
}

class ElectricPowerSwitch {
  constructor(device) {
    this.device = device
    this.isOn = false
  }

  press() {
    if (this.isOn) {
      this.device.turnOff()
      this.isOn = false
    } else {
      this.device.turnOn()
      this.isOn = true
    }
  }
}
```

Setiap contoh di atas menunjukkan cara mengimplementasikan prinsip SOLID dalam JavaScript, membantu Anda menulis kode yang lebih bersih, terorganisir, dan mudah dikelola.