Nama : Haikal Fadhilah Mufid

NIM : 2311104027

MENJELASKAN SALAH SATU DESIGN PATTERN

1. **Berikan salah satu contoh kondisi dimana design pattern “Observer” dapat digunakan**
   Observer cocok digunakan pada aplikasi notifikasi. Misalnya, ketika seorang user memposting sesuatu di media sosial, semua follower (observer) secara otomatis mendapat pemberitahuan tanpa perlu dicek satu per satu.

2. **Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Observer”**
   Buat Subject (pengirim notifikasi) yang dapat menambah/menghapus observer, dan memberi tahu mereka saat terjadi perubahan. Setiap Observer harus memiliki method yang dipanggil saat diberi tahu perubahan. Subject akan memanggil method itu untuk setiap observer yang terdaftar.

3. **Berikan kelebihan dan kekurangan dari design pattern “Observer”**

   Kelebihan :
   A. Memudahkan komunikasi antar objek tanpa harus saling mengetahui implementasi.
   B. Mendukung prinsip low coupling.
   C. Mudah menambah observer baru tanpa mengubah kode existing.

   Kekurangan :

   A. Sulit melacak dependensi saat jumlah observer banyak.
   B. Potensi masalah performa jika terlalu banyak observer.
   C. Debugging bisa rumit karena alur notifikasi tidak eksplisit.

IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN OBSERVER

```
// Notifier class to manage event subscriptions
class Notifier {
  constructor() {
    this.subscribers = {};
  }

  on(event, handler) {
    if (!this.subscribers[event]) {
      this.subscribers[event] = [];
    }
    this.subscribers[event].push(handler);
  }

  trigger(event, payload) {
    if (this.subscribers[event]) {
      this.subscribers[event].forEach(fn => fn(payload));
    }
  }
}

// TextEditor class that fires events
class TextEditor {
  constructor() {
    this.notifier = new Notifier();
    this.currentFile = null;
  }

  openFile(filename) {
    this.currentFile = { name: filename };
    this.notifier.trigger("fileOpened", this.currentFile.name);
  }

  saveFile() {
    if (this.currentFile) {
      this.notifier.trigger("fileSaved", this.currentFile.name);
    }
  }
}

// Logger class (simulates logging)
class ConsoleLogger {
  constructor(messageTemplate) {
```

```
        this.messageTemplate = messageTemplate;
    }

    log(filename) {
        console.log(this.messageTemplate.replace("%s", filename));
    }
}

// Email simulator (shows output in terminal only)
class EmailSimulator {
    constructor(recipient, messageTemplate) {
        this.recipient = recipient;
        this.messageTemplate = messageTemplate;
    }

    send(filename) {
        const message = this.messageTemplate.replace("%s", filename);
        console.log(`(Simulasi Kirim Email ke ${this.recipient}) => ${message}`);
    }
}

// Setup
const editor = new TextEditor();

const logger = new ConsoleLogger("LOG: File '%s' telah dibuka.");
const email = new EmailSimulator("admin@example.com", "PERINGATAN: File '%s' telah
disimpan!");

editor.notifier.on("fileOpened", logger.log.bind(logger));
editor.notifier.on("fileSaved", email.send.bind(email));

// Simulasi penggunaan
editor.openFile("laporan_keuangan.pdf");
editor.saveFile();
```

Kode tersebut sudah menggunakan pola Observer karena ada objek yang mengirim notifikasi (publisher) dan objek lain yang menerima notifikasi (subscriber). Saat sebuah event terjadi, publisher memberi tahu subscriber yang sudah mendaftar untuk event tersebut sehingga subscriber dapat merespon tanpa publisher perlu tahu detailnya. Ini membuat komunikasi antar objek lebih terorganisir dan fleksibel.