Tarea Final

```
class Database {
  constructor() {
    if (!Database.instance) {
      Database.instance = this;
    }
    return Database.instance;
  }

  query(sql) {
    console.log("Ejecutando consulta:", sql);
  }
}

const db1 = new Database();
const db2 = new Database();

console.log(db1 === db2); // Output: true
db1.query("SELECT * FROM users");
```

Singleton: Asegura que, a pesar de la creación de más de una clase Database, solamente exista una instancia de ésta; es decir, si ya existe una instancia, simplemente regresa la misma ya creada.

Este código tiene como función crear una sola instancia de una base de datos, y poder realizar consultas de sql dentro de la misma.

```
class Logger {
  constructor() {
    this.logs = [];
  }

  log(message) {
    this.logs.push(message);
    console.log("Log registrado:", message);
  }

  static getInstance() {
    if (!Logger.instance) {
      Logger.instance = new Logger();
    }
    return Logger.instance;
  }
}

const logger1 = Logger.getInstance();
const logger2 = Logger.getInstance();

console.log(logger1 === logger2); // Output: true
logger1.log("Error: No se puede conectar al servidor");
```

Singleton: Este archivo también demuestra el patrón singleton, ya que solamente permite la instancia de la clase Logger una vez. Esto también significa que cada logger creado a fuerzas tendrá el mismo arreglo de mensajes añadidos en la función log.

Este código tiene como propósito crear un solo log que registra una serie de mensajes dados en un arreglo.

```
class User {
  constructor(name) {
    this.name = name;
  }

  greet() {
    console.log("Hola, soy", this.name);
  }
}

class UserFactory {
  createUser(name) {
    return new User(name);
  }
}

const factory = new UserFactory();
const user1 = factory.createUser("Juan");
const user2 = factory.createUser("Maria");

user1.greet(); // Output: Hola, soy Juan
user2.greet(); // Output: Hola, soy Maria
```

Sinceramente no recuerdo haber visto este patrón en clase, pero se puede identificar el principio de single responsibility además del polimorfismo, ya que el código es reutilizado.

Este código tiene como propósito crear un objeto capaz de crear objetos de una clase en particular; en este caso son usuarios. Se crean con un nombre, y tienen una función que permite ‘saludar’ al usuario creado.