Vamos a revisar cómo se cumplen los requisitos específicos en el código que te he proporcionado para la gestión de una tienda de ropa, según los criterios que mencionaste anteriormente:

### Requisitos y Cumplimiento en el Código

1. \*\*Clases para representar prendas de vestir (camisetas, pantalones, zapatos):\*\*

- \*\*Clase Base `Prenda` y Subclases (`Camiseta`, `Pantalon`, `Zapato`):\*\*

```javascript

class Prenda {

constructor(nombre, talla, color, precio) {

this.nombre = nombre;

this.talla = talla;

this.color = color;

this.precio = precio;

}

mostrarDetalles() {

return `${this.nombre}, Talla: ${this.talla}, Color: ${this.color}, Precio: ${this.precio}`;

}

}

class Camiseta extends Prenda {

constructor(nombre, talla, color, precio, manga) {

super(nombre, talla, color, precio);

this.manga = manga;

}

mostrarDetalles() {

return `${super.mostrarDetalles()}, Manga: ${this.manga}`;

}

}

class Pantalon extends Prenda {

constructor(nombre, talla, color, precio, tipo) {

super(nombre, talla, color, precio);

this.tipo = tipo;

}

mostrarDetalles() {

return `${super.mostrarDetalles()}, Tipo: ${this.tipo}`;

}

}

class Zapato extends Prenda {

constructor(nombre, talla, color, precio, estilo) {

super(nombre, talla, color, precio);

this.estilo = estilo;

}

mostrarDetalles() {

return `${super.mostrarDetalles()}, Estilo: ${this.estilo}`;

}

}

```

\*\*Cumplimiento\*\*: Se han definido clases (`Prenda`, `Camiseta`, `Pantalon`, `Zapato`) para representar diferentes tipos de prendas de vestir. Cada clase tiene sus propiedades específicas y métodos para mostrar detalles (`mostrarDetalles()`).

2. \*\*Herencia entre las clases de prendas de vestir para aprovechar las características comunes:\*\*

- \*\*Herencia en las Clases (`Camiseta`, `Pantalon`, `Zapato`):\*\*

Las subclases (`Camiseta`, `Pantalon`, `Zapato`) extienden la clase base `Prenda`, heredando sus propiedades y métodos. Por ejemplo:

```javascript

class Camiseta extends Prenda {

// Constructor y métodos específicos para camisetas

}

class Pantalon extends Prenda {

// Constructor y métodos específicos para pantalones

}

class Zapato extends Prenda {

// Constructor y métodos específicos para zapatos

}

```

\*\*Cumplimiento\*\*: Las subclases (`Camiseta`, `Pantalon`, `Zapato`) heredan propiedades y métodos de la clase base `Prenda`, lo que permite reutilizar y extender características comunes.

3. \*\*Polimorfismo para definir métodos específicos para cada tipo de prenda:\*\*

- \*\*Métodos `mostrarDetalles()` en Subclases:\*\*

Cada subclase (`Camiseta`, `Pantalon`, `Zapato`) redefine el método `mostrarDetalles()` para incluir información específica de cada tipo de prenda.

```javascript

mostrarDetalles() {

return `${super.mostrarDetalles()}, Manga: ${this.manga}`;

}

```

\*\*Cumplimiento\*\*: Se ha implementado polimorfismo al redefinir el método `mostrarDetalles()` en cada subclase para mostrar detalles específicos de camisetas, pantalones y zapatos.

4. \*\*Sobrecarga de métodos para realizar acciones comunes con diferentes tipos de prendas:\*\*

- \*\*Función `agregarPrendaALista()` en JavaScript:\*\*

La función `agregarPrendaALista()` maneja la adición de cualquier tipo de prenda (`Camiseta`, `Pantalon`, `Zapato`) a la lista correspondiente en el inventario.

```javascript

function agregarPrendaALista(prenda) {

// Agrega la prenda a la lista correspondiente según su tipo

}

```

\*\*Cumplimiento\*\*: La función `agregarPrendaALista()` está diseñada para manejar diferentes tipos de prendas y agregarlas correctamente a listas separadas en el inventario.

5. \*\*Encapsulación para proteger los datos de las clases:\*\*

- \*\*Propiedades Privadas de Clase:\*\*

En JavaScript, no hay soporte nativo para propiedades y métodos privados como en otros lenguajes orientados a objetos. Sin embargo, se sigue el principio de encapsulación al definir propiedades como `nombre`, `talla`, `color`, `precio` como propiedades de instancia y acceder a ellas a través de métodos públicos como `mostrarDetalles()`.

\*\*Cumplimiento\*\*: Aunque JavaScript no tiene encapsulación en el sentido estricto, se ha diseñado el código para utilizar métodos públicos (`mostrarDetalles()`) para acceder a los datos de las prendas y mantener la coherencia de la aplicación.

En resumen, el código proporcionado cumple con los requisitos especificados al utilizar clases para representar diferentes tipos de prendas de vestir, implementar herencia para reutilizar características comunes, usar polimorfismo para métodos específicos de cada tipo de prenda, sobrecargar métodos para acciones comunes y seguir el principio de encapsulación dentro de las capacidades de JavaScript.