Lucía Ramírez Monje 2ASIR

PROYECTO03. HERENCIAS

ÍNDICE:

PROYECTO1EV TYPESCRIPT

Lucía Ramírez Monje 2ASIR

1. NOCIONES TEÓRICAS

¿Qué es la programación orientada a objetos (POO)?

La Programación Orientada a Objetos se define como una manera de programar específica, donde se organiza el código en unidades denominadas clases, de las cuales se crean objetos que se relacionan entre sí para conseguir los objetivos de las aplicaciones.

- Encapsulación

  La encapsulación es el empaquetamiento de datos y funciones en un componente (por ejemplo, una clase) para luego controlar el acceso a ese componente. Debido a esto, un usuario de esa clase solo necesita conocer su interfaz (es decir, los datos y las funciones expuestas fuera de la clase), no la implementación oculta.

- Herencia

  Decir que una clase hereda de otra quiere decir que esa clase obtiene los mismos métodos y propiedades de la otra clase. Permitiendo de esta forma añadir a las características heredadas las suyas propias.

- Sobreescritura
  - No hay que confundir la sobrecarga con la sobreescritura.
  - Sobrecargar significa definir nuevos métodos.
  - Sobrescribir significa ocultar un método con una nueva definición de ese mismo método.
  - La sobrecarga no implica herencia, la sobreescritura sí.

- Uso de super

  La palabra clave super es usada para acceder y llamar funciones del padre de un objeto.

- Polimorfismo

  Es la capacidad para que, al enviar el mismo mensaje (o, en otras palabras, invocar al mismo método) desde distintos objetos, cada uno de esos objetos pueda responder a ese mensaje (o a esa invocación) de forma distinta.

- Persistencia

  Se llama persistencia a la capacidad de guardar la información de un programa para poder volver a utilizarla en otro momento.

2. CONCEPTOS USADOS

- Encapsulación

  En este caso encapsulamos los componentes de la clase Vivienda y Empleado, menos Ciudad. Para referirnos a los componentes privados crearemos métodos get para poder acceder a ellos.

```
export class Empleado {
    private _id: number
    private _nombre: string
    private _sueldobase: number
    private _ventas: Array<Vivienda>
    constructor(id: number, nombre: string, sueldobase: number, ventas: Array<Vivienda>) {
        this._id = id
        this._nombre = nombre
        this._sueldobase = sueldobase
        this._ventas = ventas
    }

    get id() {
        return this._id
    }
    get nombre() {
        return this._nombre
    }
    get sueldobase() {
        return this._sueldobase
    }
```

```
export abstract class Vivienda {
    private _idVivienda: number
    private _largo: number
    private _ancho: number
    private _ubicacion: {
        municipio: string,
        ciudad: string,
        codpost: number
    }
    private _caracteristicas: {
        habitaciones: number,
        baños: number,
        ascensor: boolean,
        equipamiento: Array<string>
    }
    private _preciom2: number | null
    private _estado: { vendido: boolean, fecha: Date | null }
    constructor(idVivienda: number,
        largo: number,
        ancho: number,
        ubicacion: { municipio: string, ciudad: string, codpost: number },
        caracteristicas: { habitaciones: number, baños: number, ascensor: boolean, equipamiento: Array<string> },
        precioFinal: number | null,
        estado: { vendido: boolean, fecha: Date | null }) {
        this._idVivienda = idVivienda
        this._largo = largo
        this._ancho = ancho
        this._ubicacion = ubicacion
        this._caracteristicas = caracteristicas
        this._preciom2 = precioFinal
        this._estado = estado
    }

    get idVivienda() {
        return this._idVivienda
    }
    get largo() {
        return this._largo
    }
    get ancho() {
        return this._ancho
    }
    get ubicacion() {
        return this._ubicacion
    }
```

PROYECTO1EV TYPESCRIPT

Lucía Ramírez Monje 2ASIR

- Herencia
  - La clase padre (vivienda) tiene estos campos:

```
export abstract class Vivienda {
    private _idVivienda: number
    private _largo: number
    private _ancho: number
    private _ubicacion: {
        municipio: string,
        ciudad: string,
        codpost: number
    }
    private _caracteristicas: {
        habitaciones: number,
        baños: number,
        ascensor: boolean,
        equipamiento: Array<string>
    }
    private _preciom2: number | null
    private _estado: { vendido: boolean, fecha: Date | null }
```

  - La clase hijo (chalet) tiene los métodos del padre (vivienda) más: sjardin
  - La clase hijo (chalet) tiene los campos heredados super () más los suyos propios: Piscina, Largojardin, Anchojardin.

```
export class Chalet extends Vivienda {
    private _piscina: boolean
    private _largojardin: number
    private _anchojardin: number
    constructor(
        idVivienda: number,
        largo: number, ancho: number,
        ubicacion: {
            municipio: string,
            ciudad: string,
            codpost: number
        },
        caracteristicas: {
            habitaciones: number,
            baños: number,
            ascensor: boolean,
            equipamiento: Array<string>
        },
        precioFinal: number | null,
        estado: {
            vendido: boolean,
            fecha: Date | null
        },
        piscina: boolean,
        largojardin: number,
        anchojardin: number
    ) {
        super(idVivienda, largo, ancho, ubicacion, caracteristicas, precioFinal, estado)
        this._piscina = piscina
        this._largojardin = largojardin
        this._anchojardin = anchojardin
    }
```

PROYECTO1EV TYPESCRIPT

Lucía Ramírez Monje 2ASIR

- La clase hijo (casa) tiene los métodos del padre (vivienda) más: cochera
- La clase hijo (casa) tiene los campos heredados los suyos propios: cochera

```
export class Casa extends Vivienda {
    private _cochera: boolean
    constructor(idVivienda: number,
        largo: number,
        ancho: number,
        ubicacion: {
            municipio: string,
            ciudad: string,
            codpost: number
        },
        caracteristicas: {
            habitaciones: number,
            baños: number,
            ascensor: boolean,
            equipamiento: Array<string>
        },
        precioFinal: number | null,
        estado: {
            vendido: boolean,
            fecha: Date | null
        },
        cochera: boolean
    ) {
        super(idVivienda, largo, ancho, ubicacion, caracteristicas, precioFinal, estado)
        this._cochera = cochera;
    }
```

- Sobreescritura

  Método de la clase padre:

```
preciom2() {
    let preciom2: any
    if (this.ubicacion.ciudad == "sevilla") {
        preciom2 = 1386 * this.superficie()

    } else if (this.ubicacion.ciudad == "almeria") {
        preciom2 = 1088 * this.superficie()

    } else if (this.ubicacion.ciudad == "jaen") {
        preciom2 = 823 * this.superficie()

    } else if (this.ubicacion.ciudad == "malaga") {
        preciom2 = 2442 * this.superficie()

    } else if (this.ubicacion.ciudad == "granada") {
        preciom2 = 1375 * this.superficie()

    } else if (this.ubicacion.ciudad == "cadiz") {
        preciom2 = 1555 * this.superficie()

    } else if (this.ubicacion.ciudad == "cordoba") {
        preciom2 = 1220 * this.superficie()

    } else if (this.ubicacion.ciudad == "huelva") {
        preciom2 = 1253 * this.superficie()

    }
    this._preciom2 = preciom2
    return Math.round(preciom2)
}
```

Método de la clase hijo:

Hacemos referencia al método de la clase padre con el uso de super (superclase) y a continuación se sobrescribe cambiando su definición.

```
preciom2() {
    let preciom2 = super.preciom2()
    let preciojardin = this.m2jardin()
    preciom2 = preciom2 + preciojardin
    if (this._piscina == true) {
        preciom2 += 200
    }
    return Math.round(preciom2);
}
```

- Uso de super

```
todo() {
  let resultado: string
  resultado = `${super.todo()}, ¿Tiene piscina?: ${this._piscina}, Superficie del jardin(m2): ${this.sjardin()}, `
  return resultado
}
```

- Polimorfismo

Estamos creando un array llamado viviendas de tipo Array de Empleados. El polimorfismo aparece en el uso del método sueldo () de la clase padre, ya que según el objeto al que se refiera/apunte el campo sueldo toma un valor u otro.

```
for (let e of empleados) {
    dSchemaFijo._idEmpleado = dSchemaTemporal._idEmpleado = e.id
    dSchemaFijo._nombre = dSchemaTemporal._nombre = e.nombre
    dSchemaFijo._sueldobase = dSchemaTemporal._sueldobase = e.sueldo()
    dSchemaFijo._ventas = dSchemaTemporal._ventas = e.ventas

    if (e instanceof EMP_FIJO) {
        dSchemaFijo._tipoObjeto = "Fijo"
        dSchemaFijo._complemento = e.complemento
        dSchemaFijo._horasxdia= e.horas
        oSchema = new modeloEmpleado(dSchemaFijo)
    } else if (e instanceof EMP_TEMPORAL) {
        dSchemaTemporal._tipoObjeto = "Temporal"
        dSchemaTemporal._comisionVenta = e.comision
        oSchema = new modeloEmpleado(dSchemaTemporal)
    }
```

PROYECTO1EV TYPESCRIPT

Lucía Ramírez Monje 2ASIR

- Persistencia

El uso de la persistencia se ve en los métodos de crear nuevos empleados/viviendas, porque realizamos un proceso para que ese objeto quede persistente en una colección dentro de una base de datos.

```
export const nuevoEmpleadoFijo = async () => {
    const id = parseInt(await leerTeclado('ID '));
    const nombre = await leerTeclado('Nombre: ');
    const sueldo = parseInt(await leerTeclado('Sueldo base: '));
    const horasxdia = parseInt(await leerTeclado('Horas de trabajo '));
    let e = new EMP_FIJO(id, nombre, sueldo, [], horasxdia, 0)


    await db.conectarBD()
    let oSchema: any
    let dSchemaFijo: tEmpFijo = {
        _tipoObjeto: null,
        _idEmpleado: null,
        _nombre: null,
        _sueldobase: null,
        _ventas: null,
        _complemento: null,
        _horasxdia: null
    }
    dSchemaFijo._tipoObjeto = "Fijo"
    dSchemaFijo._idEmpleado = e.id
    dSchemaFijo._nombre = e.nombre
    dSchemaFijo._sueldobase = e.sueldo()
    dSchemaFijo._ventas = e.ventas
    dSchemaFijo._complemento = e.complemento
    dSchemaFijo._horasxdia = e.horas
    oSchema = new modeloEmpleado(dSchemaFijo)
    await oSchema.save()
    await db.desconectarBD()
}
```