PROYECTO03. HERENCIAS

ÍNDICE:

PROYECTO03. HERENCIAS

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.

2. CONCEPTOS USADOS

- Encapsulación

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

```
export class Vivienda {
  private _idVivienda: number
  private _largo: number
  private _ancho: number
  public ciudad: string
  constructor(idVivienda: number, largo: number, ancho: number, ciudad: string) {
    this._idVivienda = idVivienda
    this._largo = largo
    this._ancho = ancho
    this.ciudad = ciudad
  }
```

- Herencia
  - La clase padre (vivienda) tiene los métodos: idVivienda, largo, ancho, preciom2, superficie.
  - La clase padre (vivienda) tiene los campos: idVivienda, largo, ancho, ciudad.

  - La clase hijo (chalet) tiene los métodos del padre (viviendas) más: sjardin
  - La clase hijo (chalet) tiene los campos heredados (idVivienda, largo, ancho, ciudad) más los suyos propios: Piscina, Largojardin, Anchojardin.

  - La clase hijo (casa) tiene los métodos del padre (viviendas) más: cochera
  - La clase hijo (casa) tiene los campos heredados (idVivienda, largo, ancho, ciudad) más los suyos propios: cochera

- Sobreescritura

  Método de la clase padre:

```
preciom2(): number {
  let preciom2: number;
  //1.941€ está el metro cuadrado en Sevilla
  preciom2 = 1.941 * this.superficie()
  return 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(): number {
  let preciom2: number
  let preciosjardin: number
  preciom2 = super.preciom2()
  preciosjardin = this.sjardin() * 1941
  preciom2 = preciom2 + preciosjardin
  if (this._piscina == true) {
    preciom2 += 200
  }
  return 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 vivienda. El polimorfismo aparece en el uso del método preciom2 de la clase padre, ya que según el objeto al que se refiera/apunte el campo preciom2 toma un valor u otro.