Retos(II) – Herencia en TS

*Reto 1

Crea un sistema de gestión de empleados utilizando TypeScript. El sistema debe permitir registrar diferentes tipos de empleados y realizar operaciones específicas según el tipo de empleado.

1. Define una clase base llamada `Empleado` con las siguientes propiedades:
   - `nombre` (string): representa el nombre del empleado.
   - `edad` (number): representa la edad del empleado.
   - `salario` (number): representa el salario del empleado.
2. Crea una clase llamada `EmpleadoTiempoCompleto` que herede de `Empleado`. Esta clase debe tener una propiedad adicional:
   - `horasTrabajadas` (number): representa el número de horas trabajadas por el empleado a la semana.
3. Crea una clase llamada `EmpleadoPorHoras` que también herede de `Empleado`. Esta clase debe tener una propiedad adicional:
   - `tarifaHora` (number): representa la tarifa por hora del empleado.
4. Agrega métodos a las clases `EmpleadoTiempoCompleto` y `EmpleadoPorHoras` para calcular el salario total de cada empleado.
   - En el caso de `EmpleadoTiempoCompleto`, el salario total se calcula multiplicando las horas trabajadas por la tarifa por hora.
   - En el caso de `EmpleadoPorHoras`, el salario total se calcula multiplicando las horas trabajadas por la tarifa por hora, y se le suma un bono del 10% si el empleado ha trabajado más de 40 horas a la semana.

5. Crea instancias de los diferentes tipos de empleados y muestra por consola su nombre, edad y salario total.

Recuerda utilizar TypeScript para declarar los tipos de datos de las propiedades y los métodos, así como para realizar las validaciones necesarias.

*Reto 2

Crea un sistema de gestión de figuras geométricas utilizando TypeScript. El sistema debe permitir calcular el área y el perímetro de diferentes tipos de figuras geométricas.

1. Define una clase base llamada `Figura` con los siguientes métodos:
   - `calcularArea(): number`: calcula y devuelve el área de la figura.
   - `calcularPerimetro(): number`: calcula y devuelve el perímetro de la figura.
2. Crea una clase llamada `Rectangulo` que herede de `Figura`. Esta clase debe tener las siguientes propiedades adicionales:
   - `base` (number): representa la base del rectángulo.
   - `altura` (number): representa la altura del rectángulo.
3. Crea una clase llamada `Triangulo` que también herede de `Figura`. Esta clase debe tener las siguientes propiedades adicionales:
   - `lado1` (number): representa el primer lado del triángulo.
   - `lado2` (number): representa el segundo lado del triángulo.
   - `lado3` (number): representa el tercer lado del triángulo.
4. Implementa los métodos `calcularArea()` y `calcularPerimetro()` en las clases `Rectangulo` y `Triangulo` según las fórmulas correspondientes.
5. Crea instancias de rectángulos y triángulos con diferentes dimensiones y muestra por consola su área y perímetro.

Recuerda utilizar TypeScript para declarar los tipos de datos de las propiedades y los métodos, así como para realizar las validaciones necesarias.

*Reto 3

Crea un sistema de gestión de vehículos utilizando TypeScript. El sistema debe permitir almacenar información sobre diferentes tipos de vehículos y realizar operaciones específicas según el tipo de vehículo.

1. Define una clase base llamada `Vehiculo` con las siguientes propiedades:
   - `marca` (string): representa la marca del vehículo.
   - `modelo` (string): representa el modelo del vehículo.
   - `anio` (number): representa el año de fabricación del vehículo.
   - `precio` (number): representa el precio del vehículo.
2. Crea una clase llamada `Automovil` que herede de `Vehiculo`. Esta clase debe tener una propiedad adicional:
   - `cantidadPuertas` (number): representa la cantidad de puertas del automóvil.
3. Crea una clase llamada `Motocicleta` que también herede de `Vehiculo`. Esta clase debe tener una propiedad adicional:
   - `cilindrada` (number): representa la cilindrada de la motocicleta.
4. Agrega métodos a las clases `Automovil` y `Motocicleta` para calcular el impuesto a pagar por cada vehículo.
   - En el caso de `Automovil`, el impuesto se calcula como el 2% del precio del automóvil.
   - En el caso de `Motocicleta`, el impuesto se calcula como el 1% del precio de la motocicleta.
5. Crea instancias de automóviles y motocicletas con diferentes características y muestra por consola su marca, modelo, año y el impuesto a pagar.

Recuerda utilizar TypeScript para declarar los tipos de datos de las propiedades y los métodos, así como para realizar las validaciones necesarias.

*Reto 4

Crea un sistema de gestión de personas y estudiantes utilizando TypeScript. El sistema debe permitir almacenar información sobre personas y estudiantes, y realizar operaciones específicas para cada tipo.

1. Define una clase base llamada `Persona` con las siguientes propiedades:
   - `nombre` (string): representa el nombre de la persona.
   - `edad` (number): representa la edad de la persona.
2. Crea una clase llamada `Estudiante` que herede de `Persona`. Esta clase debe tener una propiedad adicional:
   - `numeroEstudiante` (string): representa el número de estudiante.
3. Crea una clase llamada `EstudianteUniversitario` que herede de `Estudiante`. Esta clase debe tener una propiedad adicional:
   - `carrera` (string): representa la carrera que está estudiando el estudiante universitario.
4. Agrega un método `mostrarInformacion()` a cada clase para mostrar por consola la información correspondiente.
   - En el caso de `Persona`, muestra el nombre y la edad.
   - En el caso de `Estudiante`, muestra el nombre, la edad y el número de estudiante.
   - En el caso de `EstudianteUniversitario`, muestra el nombre, la edad, el número de estudiante y la carrera.

5. Crea instancias de personas, estudiantes y estudiantes universitarios con diferentes características y llama al método `mostrarInformacion()` para cada uno de ellos.

Recuerda utilizar TypeScript para declarar los tipos de datos de las propiedades y los métodos, así como para realizar las validaciones necesarias.