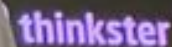
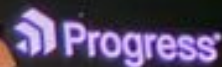




Curso de
**Fundamentos
de TypeScript**

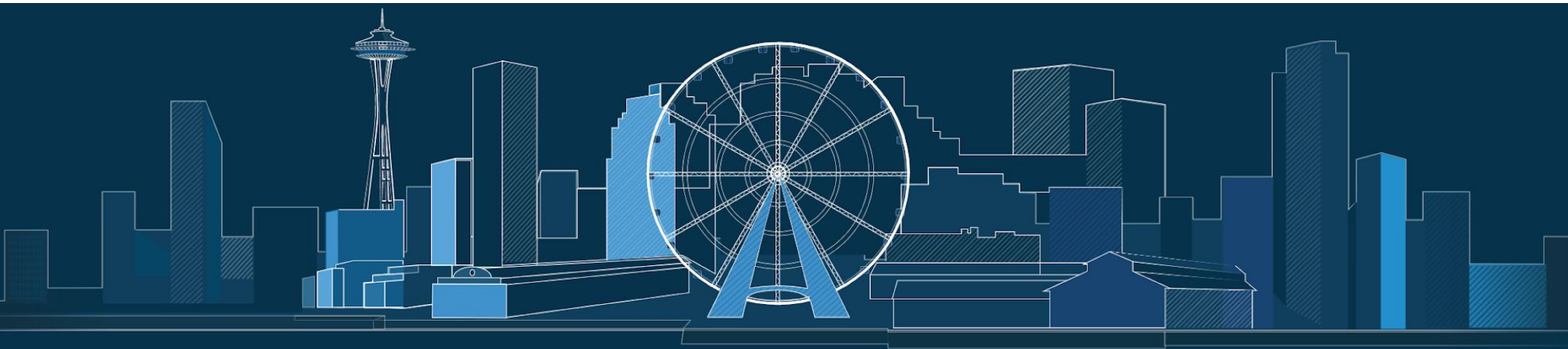
Luis Aviles
@luixaviles

WHAT IS A BLACK HOLE?



El Lenguaje de Programación TypeScript

¿Qué es TypeScript?



TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.

typescriptlang.org



¿Qué es TypeScript?

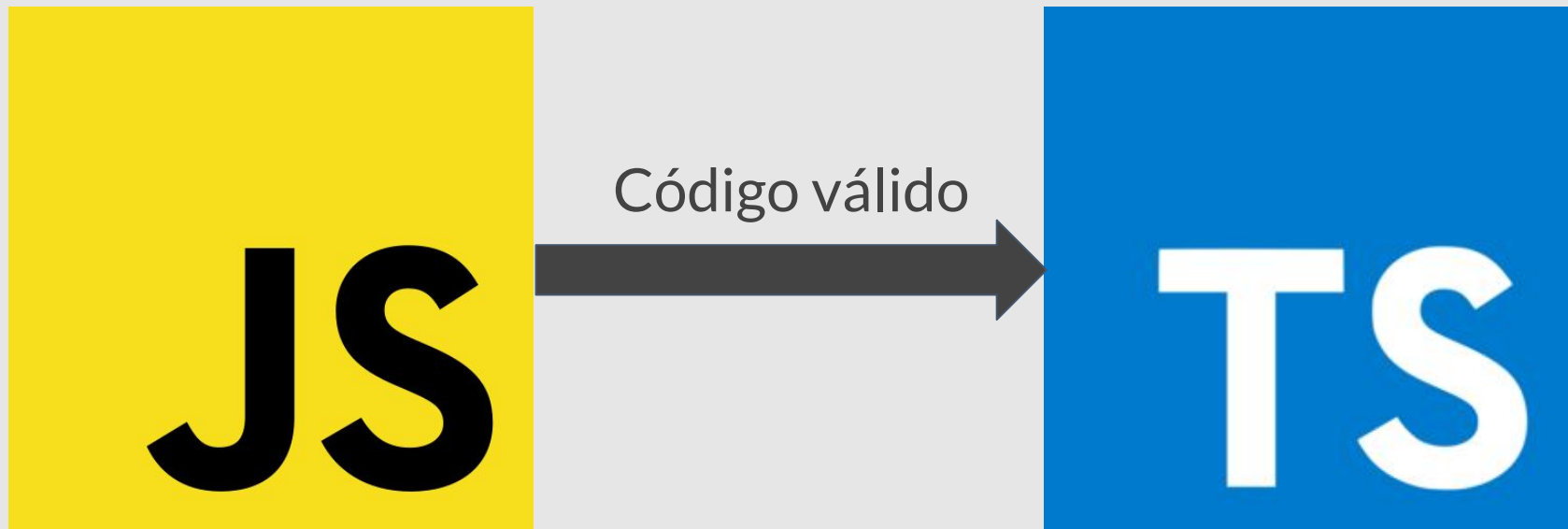
- Lenguaje de Programación Tipado
- Lenguaje de Alto Nivel
- Genera como resultado código JavaScript



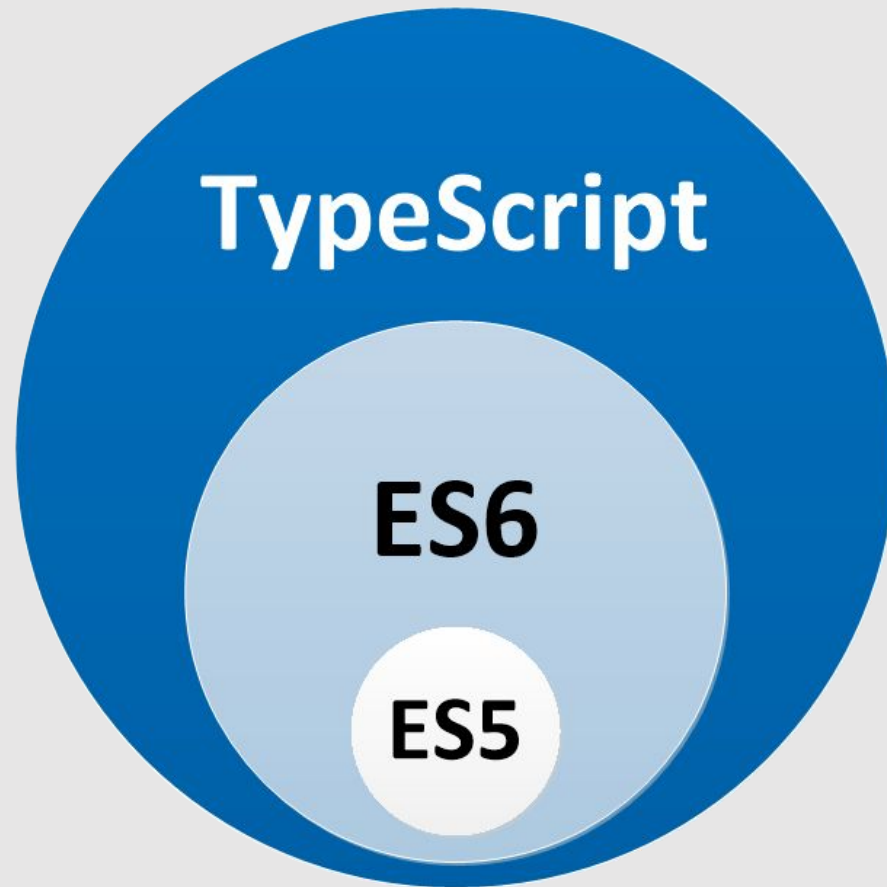
¿Qué es TypeScript?

- Código Abierto
- Desarrollo en cualquier Sistema Operativo
- El código puede ejecutarse en cualquier navegador o plataforma

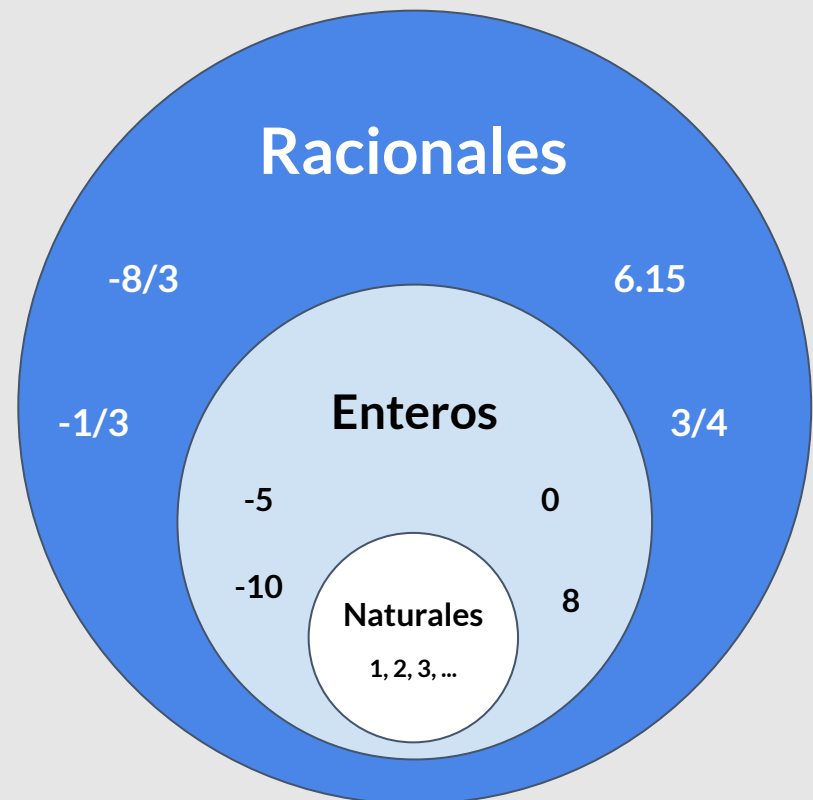
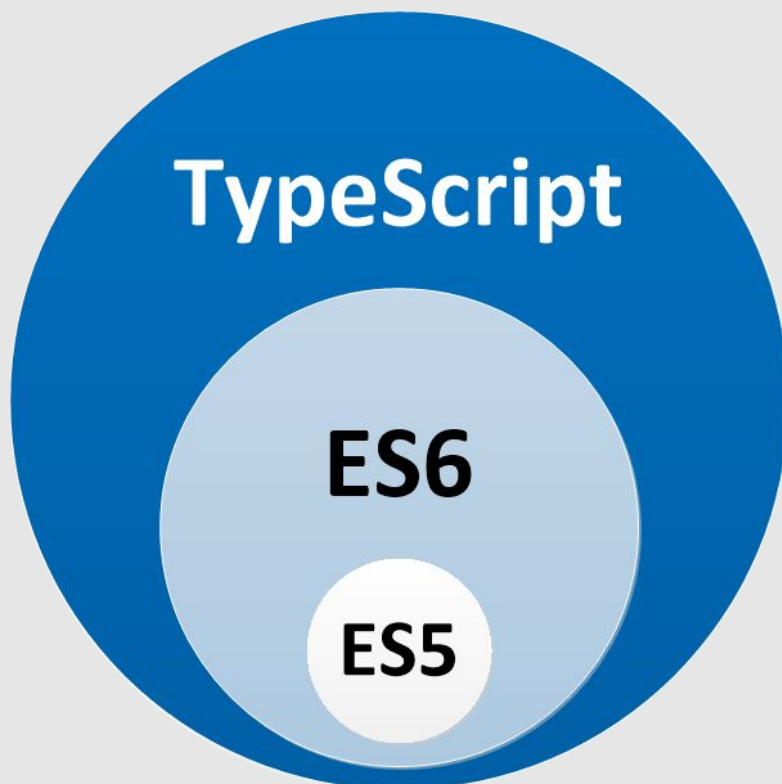
¿Qué es TypeScript?



¿Qué es TypeScript?



¿Qué es TypeScript?



¿Quién usa TypeScript?

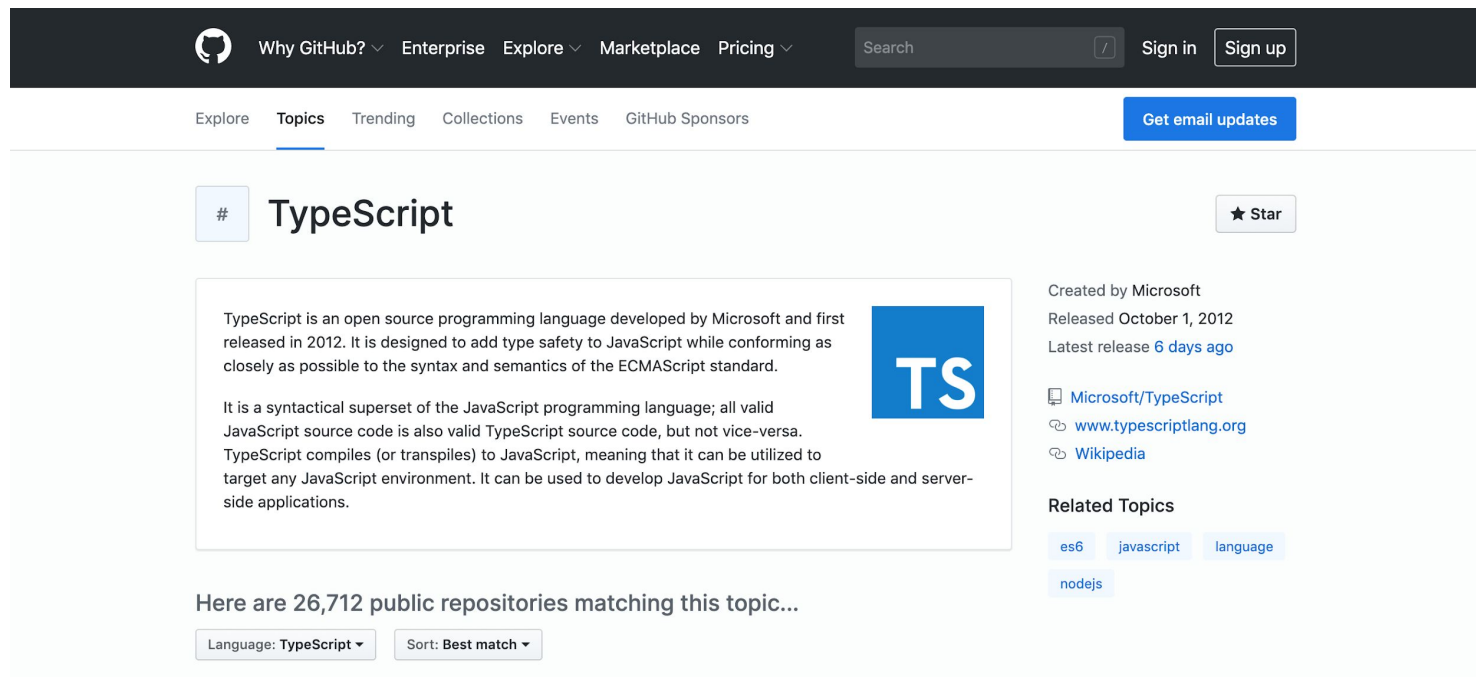


Medium

2393+ compañías

<https://stackshare.io/typescript>

¿Quién usa TypeScript?



The screenshot shows the GitHub Topics page for TypeScript. At the top is the GitHub navigation bar with links like 'Why GitHub?', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. Below this is a secondary navigation bar with 'Explore', 'Topics' (highlighted), 'Trending', 'Collections', 'Events', and 'GitHub Sponsors'. A search bar and 'Sign in'/'Sign up' buttons are also present. The main content area features a '# TypeScript' topic card. The card includes a description: 'TypeScript is an open source programming language developed by Microsoft and first released in 2012. It is designed to add type safety to JavaScript while conforming as closely as possible to the syntax and semantics of the ECMAScript standard.' It also mentions that TypeScript is a superset of JavaScript and compiles to JavaScript. To the right of the text is a blue square with 'TS' in white. Below the description, it states 'Here are 26,712 public repositories matching this topic...'. On the right side of the card, there is a '★ Star' button, a list of links (Microsoft/TypeScript, www.typescriptlang.org, Wikipedia), and a 'Related Topics' section with tags for 'es6', 'javascript', 'language', and 'nodejs'.

TypeScript

TypeScript is an open source programming language developed by Microsoft and first released in 2012. It is designed to add type safety to JavaScript while conforming as closely as possible to the syntax and semantics of the ECMAScript standard.

It is a syntactical superset of the JavaScript programming language; all valid JavaScript source code is also valid TypeScript source code, but not vice-versa. TypeScript compiles (or transpiles) to JavaScript, meaning that it can be utilized to target any JavaScript environment. It can be used to develop JavaScript for both client-side and server-side applications.

Created by Microsoft
Released October 1, 2012
Latest release 6 days ago

[Microsoft/TypeScript](#)
www.typescriptlang.org
[Wikipedia](#)

Related Topics

es6 javascript language nodejs

Here are 26,712 public repositories matching this topic...

Language: TypeScript Sort: Best match

26000+ repositorios públicos

<https://github.com/topics/typescript>



¿Por qué usar TypeScript?

- Programación Orientada a Objetos
- Potencia tu código JavaScript
- Mayor Productividad
- Poderoso sistema de tipos
- Compila a ES5, ES6 y más



¿Por qué usar TypeScript?

- Proyecto muy activo/*Open Source*
- Actualizaciones periódicas[1]
- Comunidad creciente

[1] <https://github.com/microsoft/TypeScript/wiki/Roadmap>

¿Por qué usar TypeScript?

- Puede prevenir cerca del 15% de bugs[1]

To Type or Not to Type: Quantifying Detectable Bugs in JavaScript

Zheng Gao
University College London
London, UK
z.gao.12@ucl.ac.uk

Christian Bird
Microsoft Research
Redmond, USA
cbird@microsoft.com

Earl T. Barr
University College London
London, UK
e.barr@ucl.ac.uk

http://ttendency.cs.ucl.ac.uk/projects/type_study/documents/type_study.pdf

¿Por qué usar TypeScript?

Cliente

Servidor

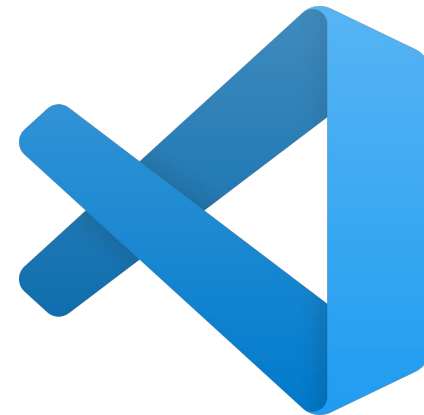


TS



Instalación de Herramientas

Instalación de Herramientas



Instalación de Herramientas

Windows



Instalación de Herramientas

macOS

Navegación y Refactorización

El Compilador de TypeScript

tsc: *TypeScript Compiler*

Instalación de tsc

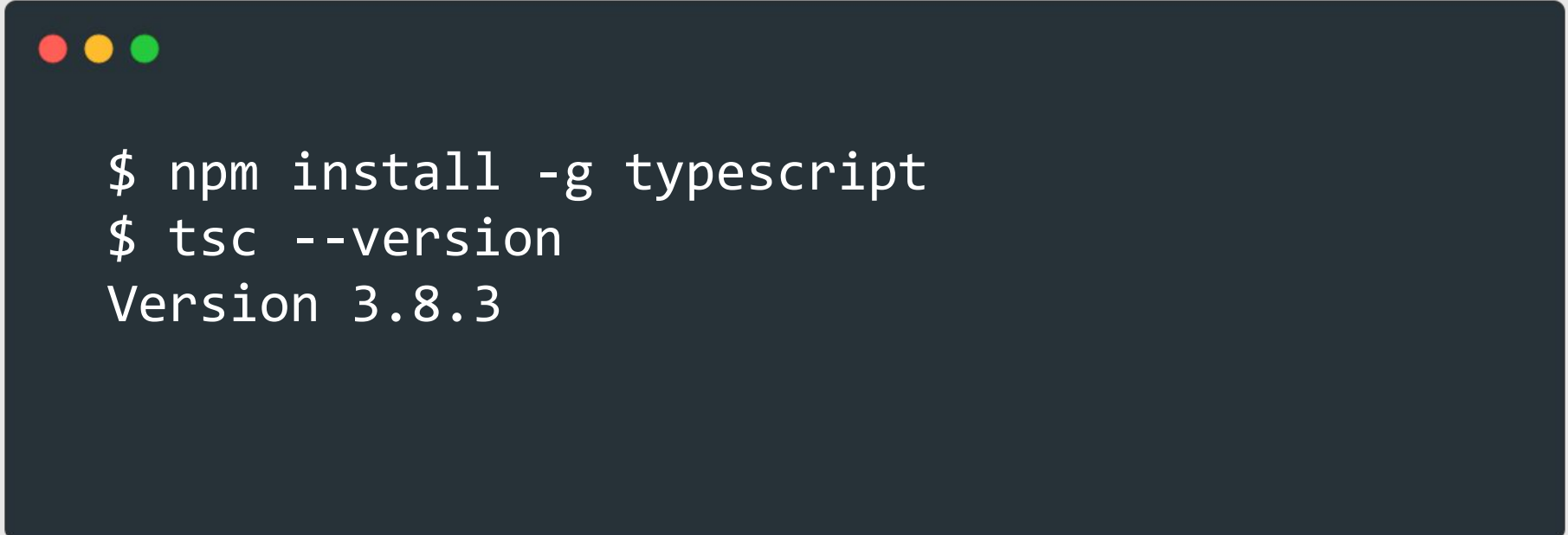
```
$ npm install -g typescript
```

Instalación de tsc Verificación



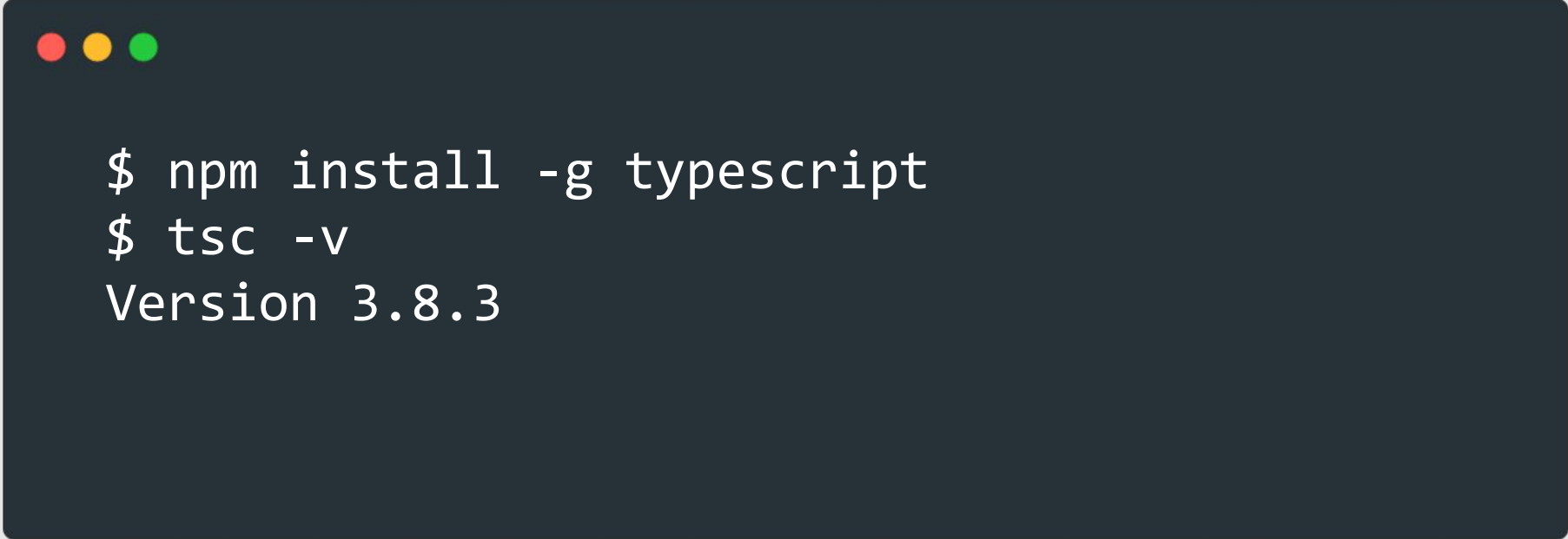
```
$ npm install -g typescript  
$ tsc --version
```


Instalación de tsc Verificación



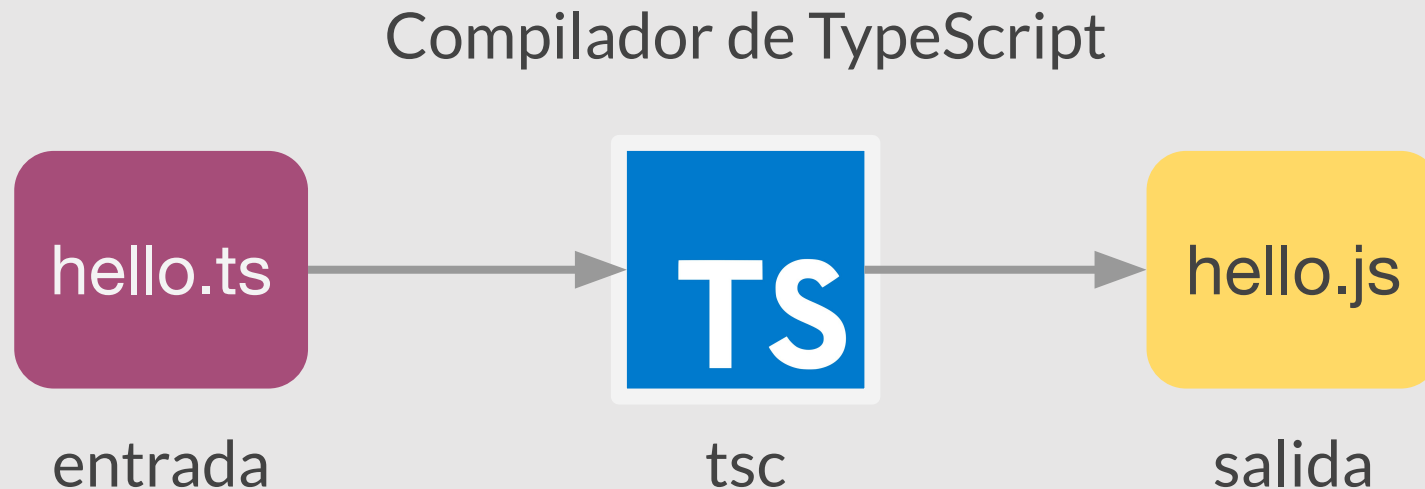
```
$ npm install -g typescript  
$ tsc --version  
Version 3.8.3
```

Instalación de tsc Verificación



```
$ npm install -g typescript  
$ tsc -v  
Version 3.8.3
```

Usando el compilador tsc

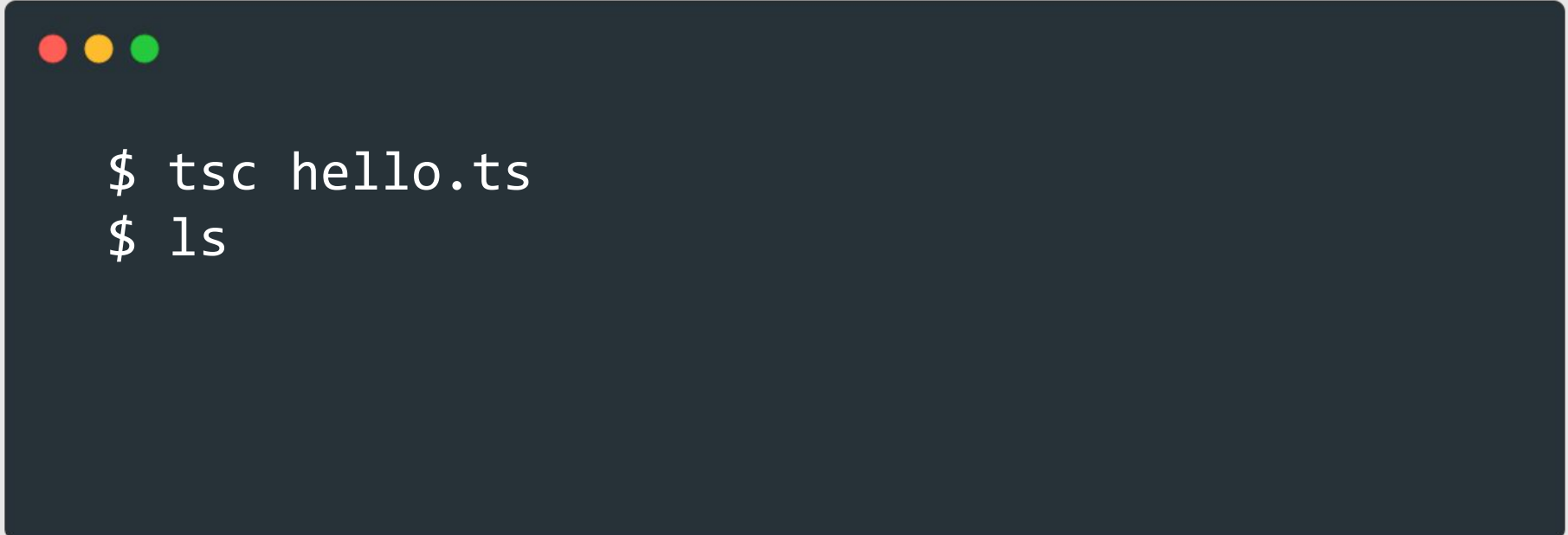


Usando el compilador tsc



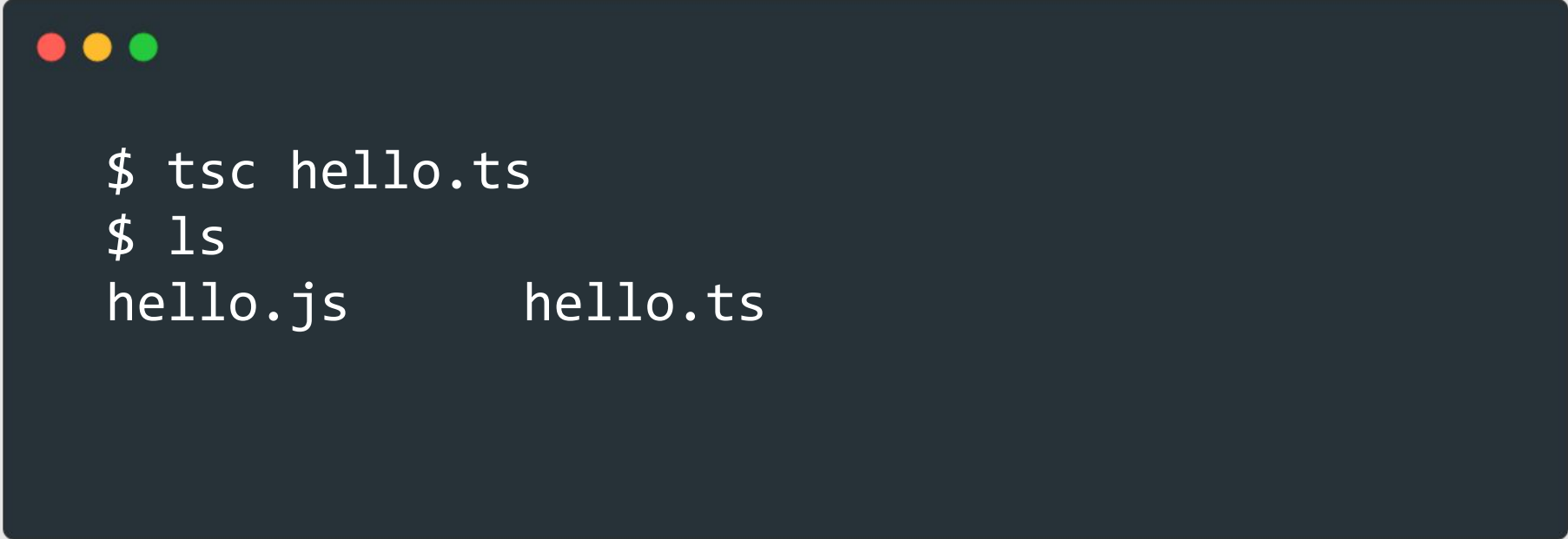
```
$ tsc hello.ts
```

Usando el compilador tsc



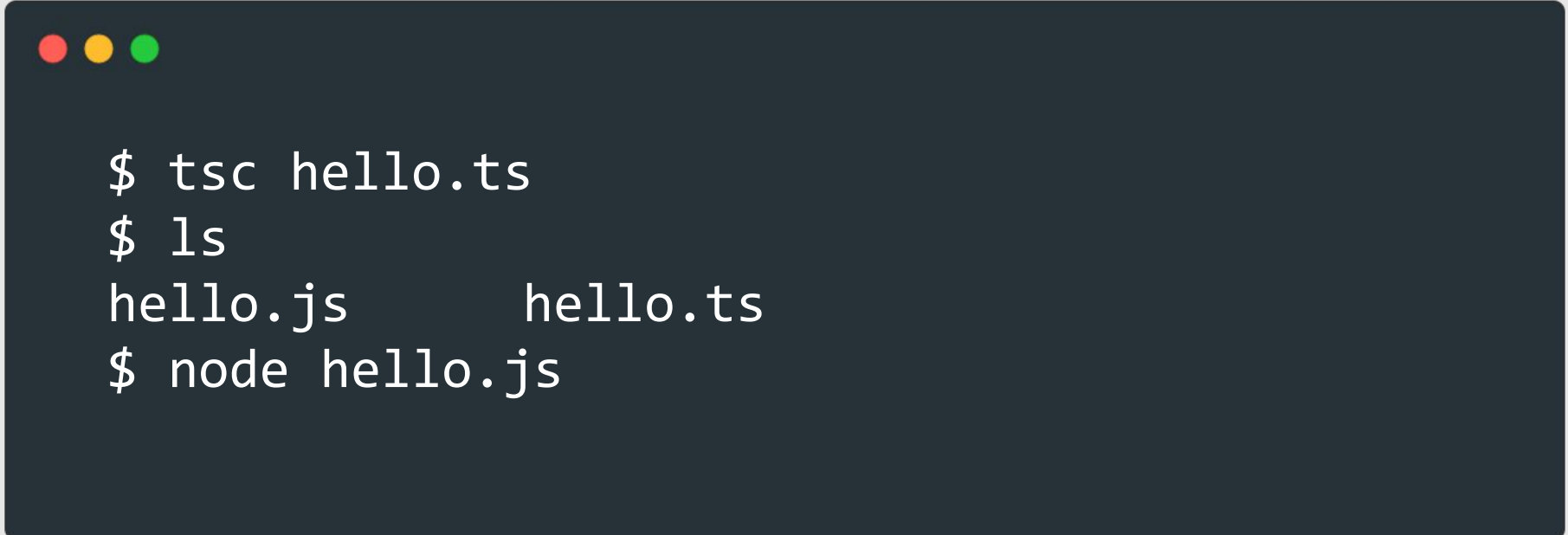
```
$ tsc hello.ts  
$ ls
```

Usando el compilador tsc



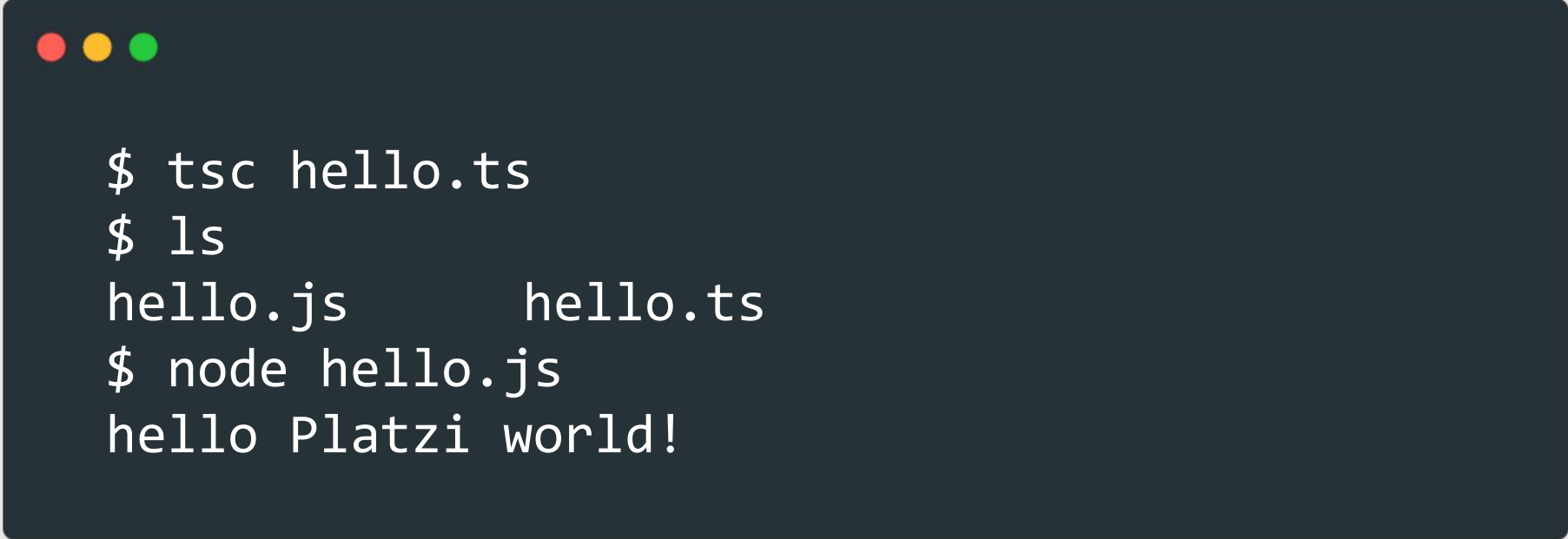
```
$ tsc hello.ts  
$ ls  
hello.js      hello.ts
```

Usando el compilador tsc



```
$ tsc hello.ts  
$ ls  
hello.js      hello.ts  
$ node hello.js
```

Usando el compilador tsc



```
$ tsc hello.ts
$ ls
hello.js      hello.ts
$ node hello.js
hello Platzi world!
```


Usando el compilador tsc

La opción --watch



```
$ tsc --watch hello.ts
```

El Archivo de Configuración de TypeScript

tsconfig.json

Archivo de Configuración tsconfig.json

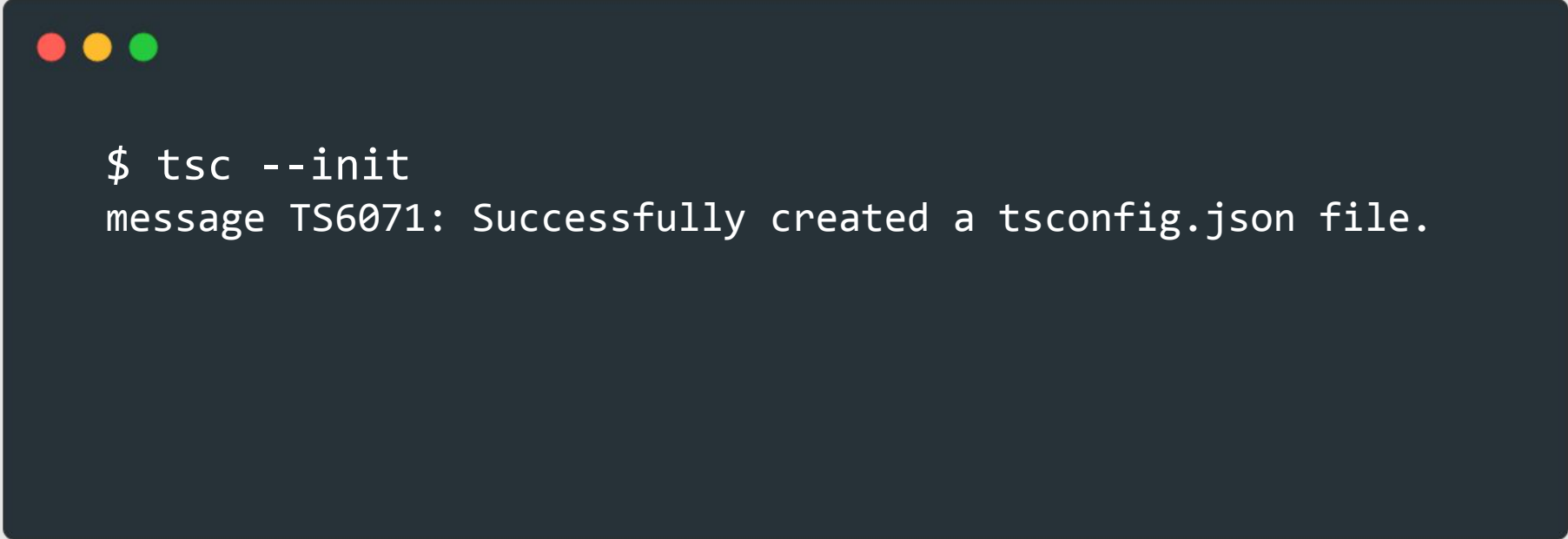
- Especifica la raíz de un proyecto TypeScript
- Permite configurar opciones para el compilador

Generación del archivo tsconfig.json



```
$ tsc --init
```

Generación del archivo tsconfig.json



```
$ tsc --init  
message TS6071: Successfully created a tsconfig.json file.
```

tsconfig.json

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "module": "commonjs",  
    "strict": true,  
    "removeComments": true  
  }  
}
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "strict": true,
    "removeComments": true
  },
  "include": [
    "src/**/*.ts"
  ]
}
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "strict": true,
    "removeComments": true
  },
  "include": [
    "src/**/*.ts"
  ],
  "exclude": [
    "node_modules",
    "**/*.test.ts"
  ]
}
```


tsconfig.json

```
{
  "extends": "./configs/base",
  "compileOnSave": true,
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "strict": true,
    "removeComments": true
  },
  "include": [
    "src/**/*.ts"
  ],
  "exclude": [
    "node_modules",
    "**/*.test.ts"
  ]
}
```

Usando el archivo tsconfig.json



```
$ tsc
```

```
$ tsc --project platzi
```

```
$ tsc file.ts
```

Usando el archivo tsconfig.json

\$ tsc → Busca la configuración

\$ tsc --project platzi → Especifica un directorio que contiene la configuración

\$ tsc file.ts → Omite la configuración

Mi Primer Proyecto TypeScript

[Overview](#)[Discussions](#)[Photos](#)[Members](#)[Map](#)



Tipado en TypeScript



Tipado en TypeScript

- **Explícito**

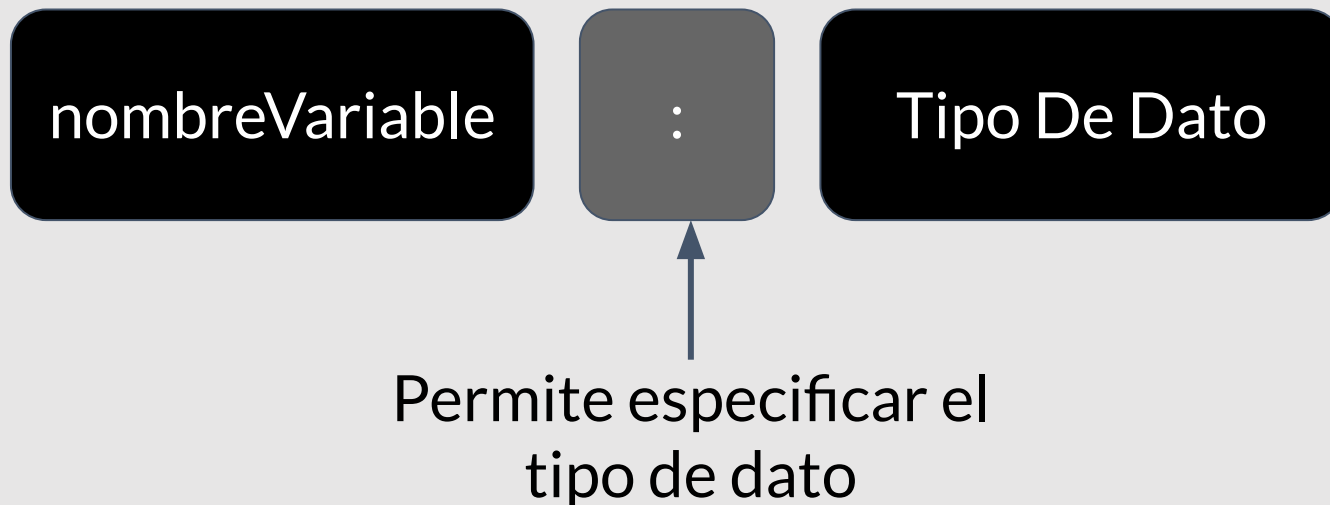
Define una sintaxis para la creación de variables con tipo de dato.

- **Inferido**

TypeScript tiene la habilidad de “deducir” el tipo en función de un valor.

Tipado en TypeScript

Explícito



Tipado en TypeScript

Inferido

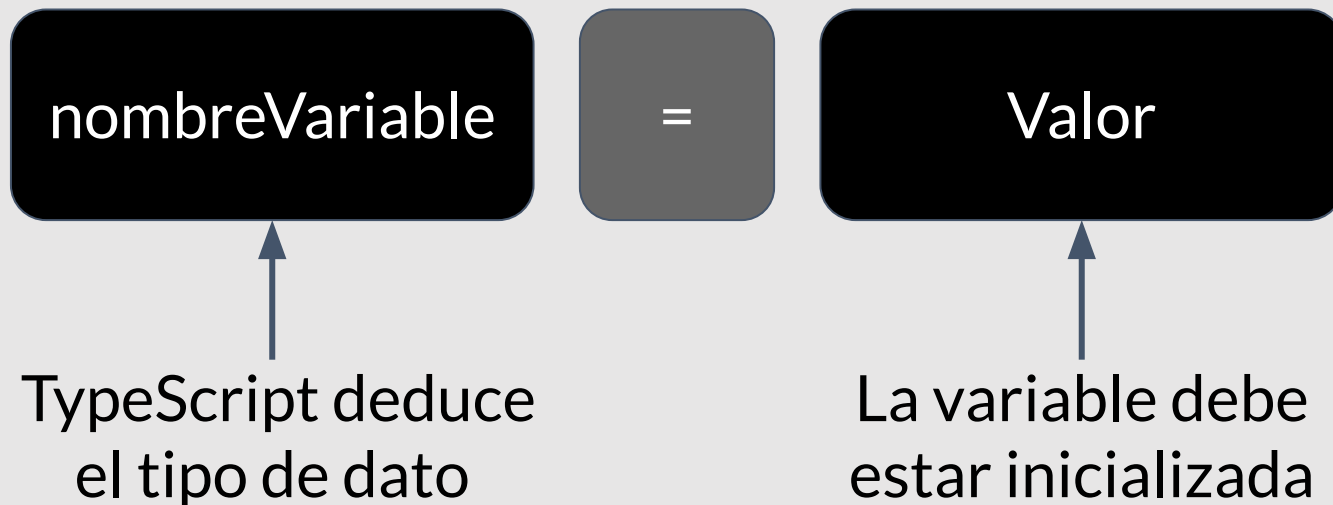
nombreVariable

=

Valor

Tipado en TypeScript

Implícito





Tipos Básicos

Tipos para datos simples
en TypeScript

Tipos Primitivos

Number

Boolean

String

Array

Tuple

Enum

Any

Void

Null

Undefined

Never

Object

Number, Boolean y String



Tipo: Number

- Valores numéricos
- Valores hexadecimales
- Valores binarios
- Valores octales



Tipo: Boolean

- El tipo de dato más simple en TypeScript
- Dos únicos valores: `true` y `false`



Tipo: String


- El tipo de dato para trabajar con datos textuales o cadenas
- Así como en JavaScript, se pueden usar comillas dobles (") y simples (')

Tipo: String

Template Strings

- Permiten definir múltiples líneas de texto
- Pueden contener expresiones o variables embebidas
- Se debe usar el caracter backtick/backquote(`) y para expresiones `${expr}`

Any, Void y Never



Tipo: Any

- Usado para capturar valores dinámicos
- Los valores pueden cambiar de tipo en el tiempo:
 - APIs externas
 - Librerías de terceros



Tipo: Void

- **void** es lo opuesto de **any**:
representa la ausencia de tipo
- Comúnmente se usa como
tipo de retorno en funciones



Tipo: Never

- Representa el tipo de valor que nunca ocurre
 - Funciones que lanzan excepciones
 - Funciones que nunca retornan un valor



Null y Undefined

Tipo: Null y Undefined

- En TypeScript, ambos “valores” tienen sus respectivos tipos:
 - `null`
 - `undefined`

Tipo: Null y Undefined Como subtipos

- Null y Undefined se pueden asumir como subtipos de los otros tipos de datos.
- Significa que se puede asignar `null` y `undefined` a una variable de tipo `string`, por ejemplo.

Tipo: Null y Undefined

La Opción `--strictNullChecks`

- Solo permite asignar `null` y `undefined` a una variable de tipo `any` o sus tipos respectivos
- Ayuda a evitar errores comunes en programación de apps en el ámbito JavaScript

Tipo: Null y Undefined

La Opción --strictNullChecks



```
$ tsc --watch src/main.ts --strictNullChecks
```

```
// Archivo tsconfig.json
```

```
"strict": true
```

```
"strict": false
```

```
"strictNullChecks": true
```

```
"strictNullChecks": false
```



Object



Tipo: object

- **object** es el tipo de dato que representa un valor no primitivo
- Es el tipo para variable que no sea number, string, boolean, null, undefined, etc.



Object vs object

- **Object:** instancia de la clase Object de JavaScript
- **object:** tipo para valores no primitivos
Con este tipo no se puede acceder a las propiedades del objeto



Array y Tuple

Array

- Al igual que JavaScript, TypeScript permite definir un arreglo para contener un conjunto de valores
- Usa dos notaciones: `[]` y `Array<tipo>`



Tuple

- Las tuplas permiten expresar un arreglo con un número fijo de elementos
- Los tipos de datos son conocidos



Enum



Enum

- Los enumerados permiten definir un conjunto de constantes con nombre
- Tienen la ventaja de adaptarse al contexto de la aplicación

Unión de Tipos, Alias y Tipos Literales

Unión de Tipos

- En TypeScript se puede definir una variable con múltiples tipos de datos: *Union Type*
- Se usa el símbolo de *pipe* ('|') entre los tipos

Alias de Tipos

- TypeScript permite crear un alias como nuevo nombre para un tipo
- El alias se puede aplicar también a un conjunto o combinación de tipos
- Se usa la palabra reservada `type`

Tipos Literales

- Una variable con un tipo literal puede contener únicamente una cadena del conjunto
- Se usan cadenas como “tipos”, combinados con el símbolo de *pipe* (“|”) entre ellos



Aserciones de Tipo



Aserciones de Tipo

- Cuando el programador puede conocer más que TypeScript sobre el valor de una variable
- Es un mensaje al compilador: “Confía en mí, sé lo que hago”

Aserciones de Tipo

- Se parece al casting de tipos en otros lenguajes de programación
- Usa dos sintaxis: `<Angle Bracket>` y `(variable as tipo)`

Funciones en TypeScript

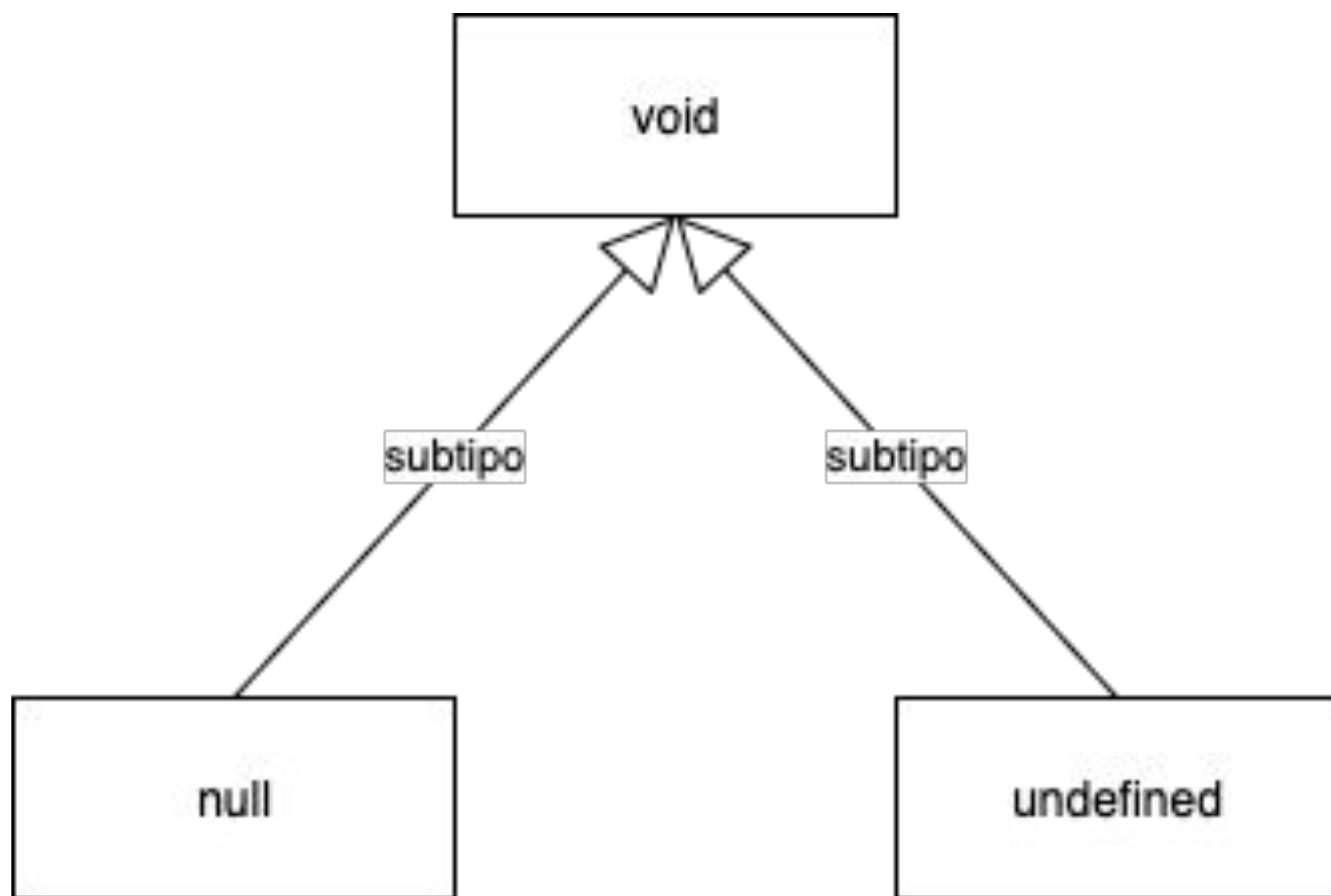


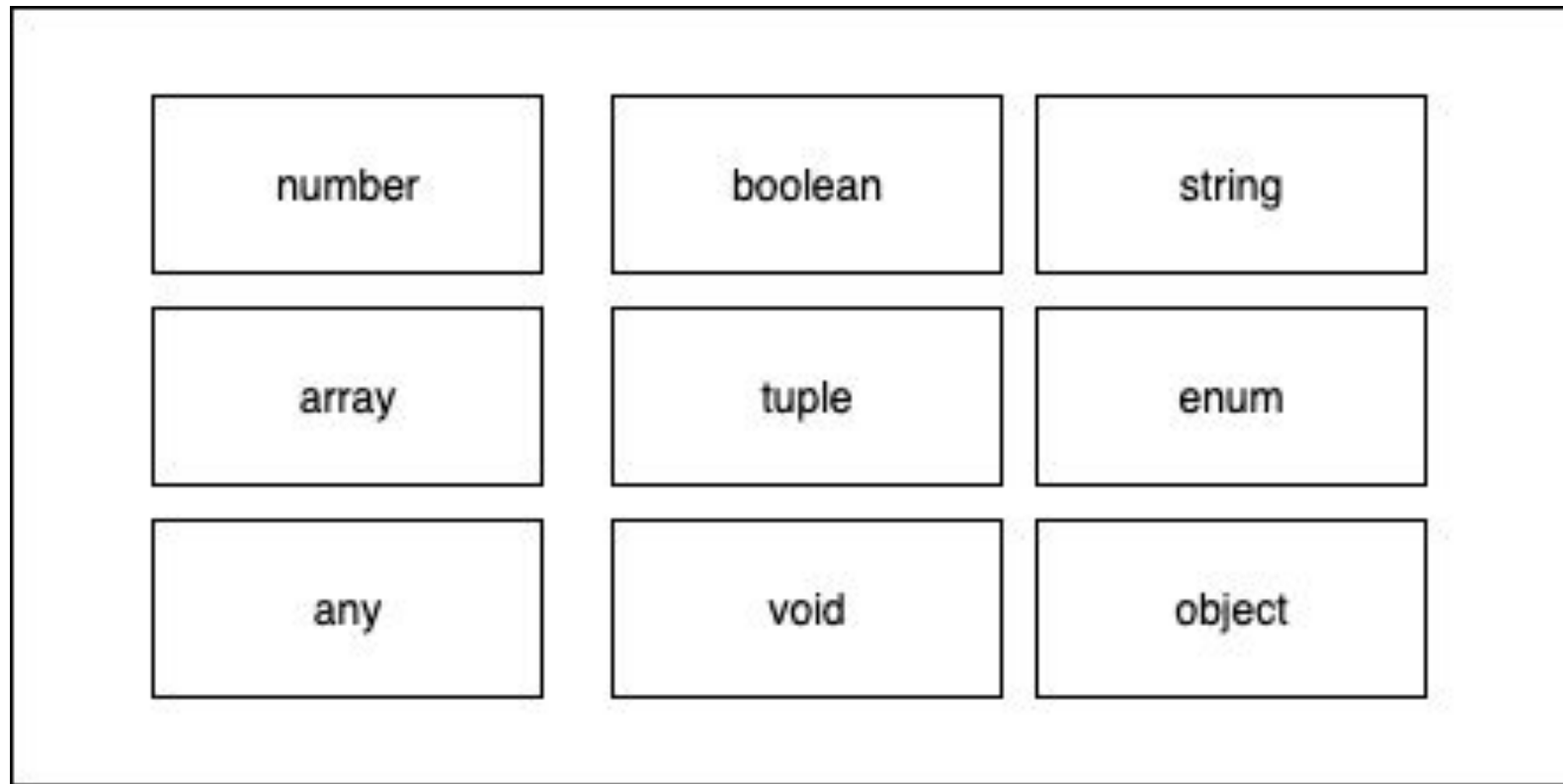
Funciones en TypeScript

- Los parámetros en las funciones son tipados
- Se pueden definir parámetros opcionales
- El tipo de retorno puede ser un tipo básico, enum, alias, tipo literal o una combinación de ellos



Resumen





`--strictNullChecks`





Interfaces



Entendiendo las Interfaces

Las Interfaces en TypeScript constituyen una forma poderosa de definir “contratos” tanto para tu proyecto, como para el código externo del mismo.

User

Album

Picture

«enum» PhotoOrientation

Interfaces

Propiedades Opcionales

No todas las propiedades de una interfaz podrían ser requeridas.

Establecemos una propiedad como opcional con el símbolo (?) después del nombre.

Interfaces

Propiedades de Solo Lectura

Algunas propiedades de la interfaz podrían no ser *modificables* una vez creado el objeto.

Esto es posible usando **readonly** antes del nombre de propiedad.



Extendiendo Interfaces

Las interfaces pueden extenderse unas de otras. Esto permite copiar los miembros ya definidos en una interfaz a otra, ganando flexibilidad y reusabilidad de componentes.



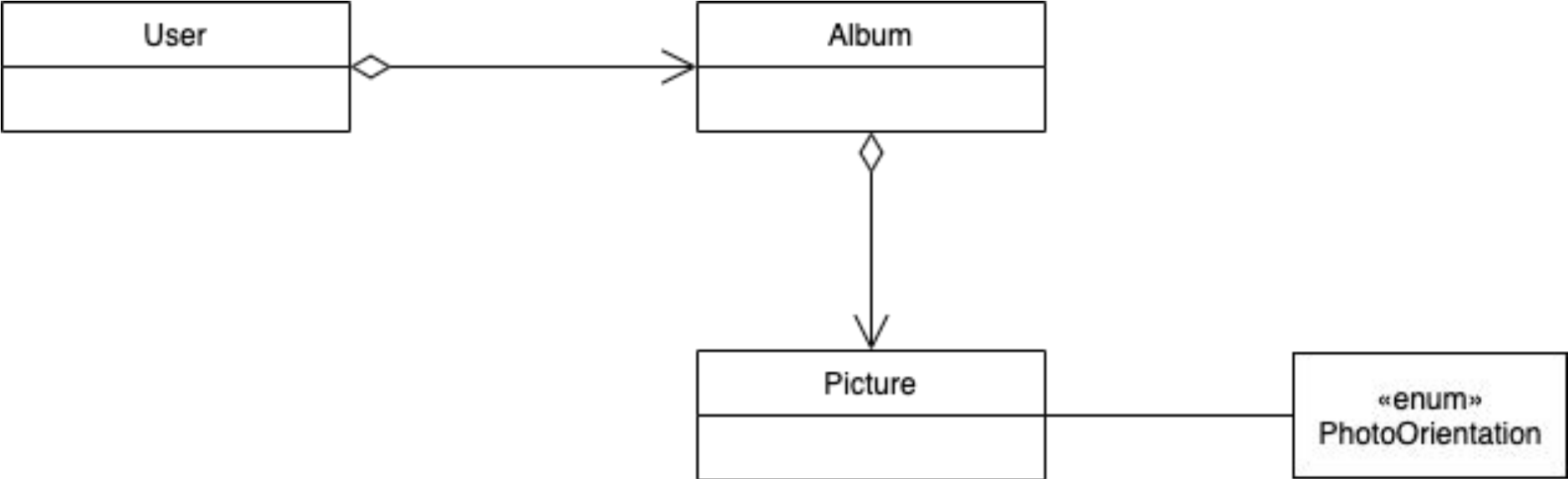
Classes

User

Album

Picture

«enum» PhotoOrientation



Definiendo Clases y Constructores

A partir de ECMAScript 2015 es posible construir clases y hacer uso del paradigma de la Programación Orientada a Objetos en JavaScript.

TypeScript permite aplicar estas técnicas sin tener que esperar por otra versión.

Clases

Miembros Públicos

TypeScript define un modificador de acceso público por defecto para los miembros de clase.

También es posible marcar un miembro como público usando la palabra reservada **public**

Clases

Miembros Privados

TypeScript define una manera propia de declarar o marcar un miembro como privado usando la palabra reservada `private`

Miembros Privados ECMAScript

TypeScript también soporta(a partir de la versión 3.8) la nueva sintaxis JavaScript para miembros privados: `#atributo`

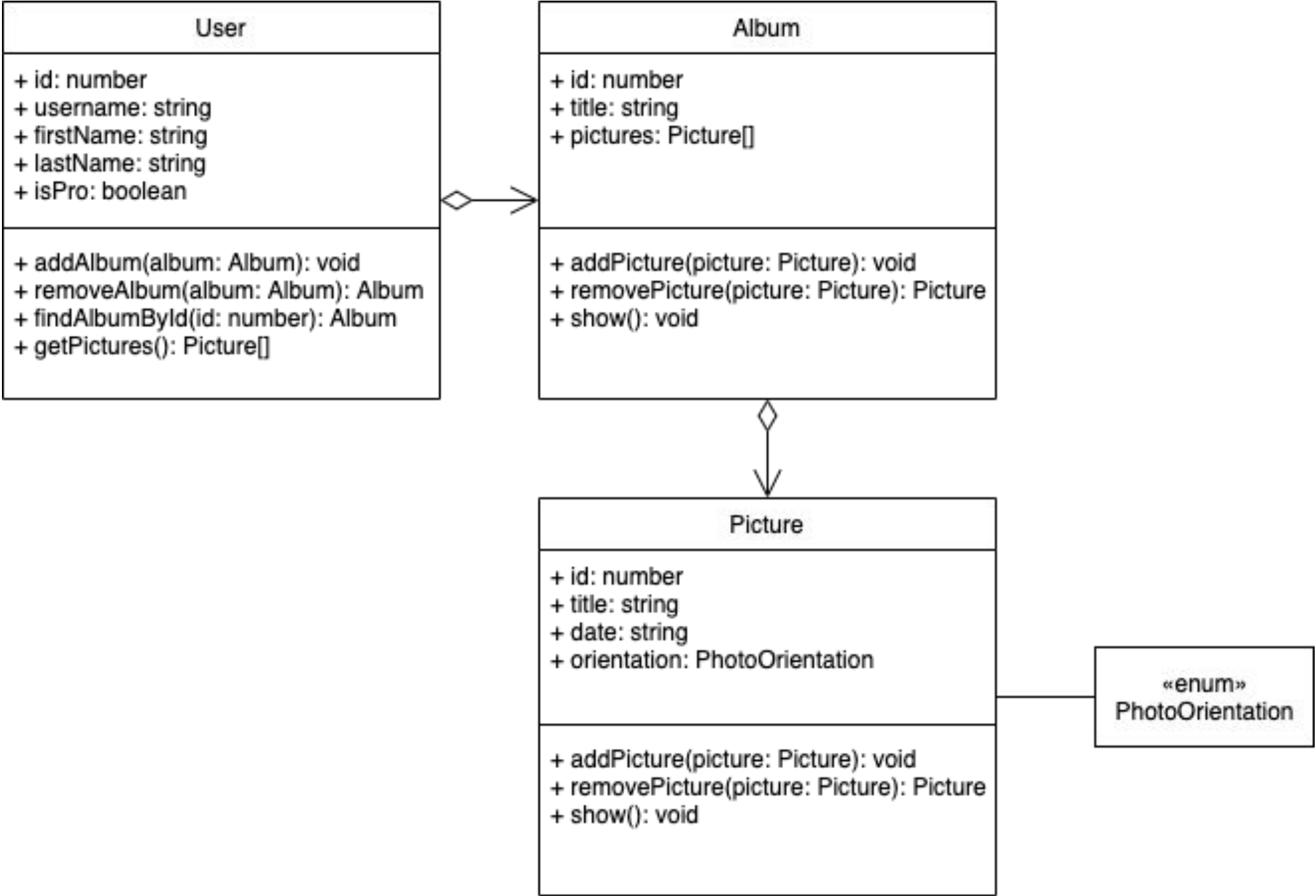
Esta característica puede ofrecer mejores garantías de aislamiento en miembros privados.



Clases

Métodos Set y Get

TypeScript soporta los métodos accesorios `set` y `get` como una forma de interceptar los accesos a los miembros privados de un objeto.



Herencia de Clases y Miembros Protegidos

TypeScript soporta este patrón común en el mundo de la POO.

Implementa la habilidad de extender código de clases existentes a través de la herencia.



Clases Abstractas

Las clases Abstractas son la base de donde otras clases podrían derivarse. A diferencia de una Interfaz, una clase abstracta puede implementar funciones para sus instancias.

La palabra reservada es: **abstract**

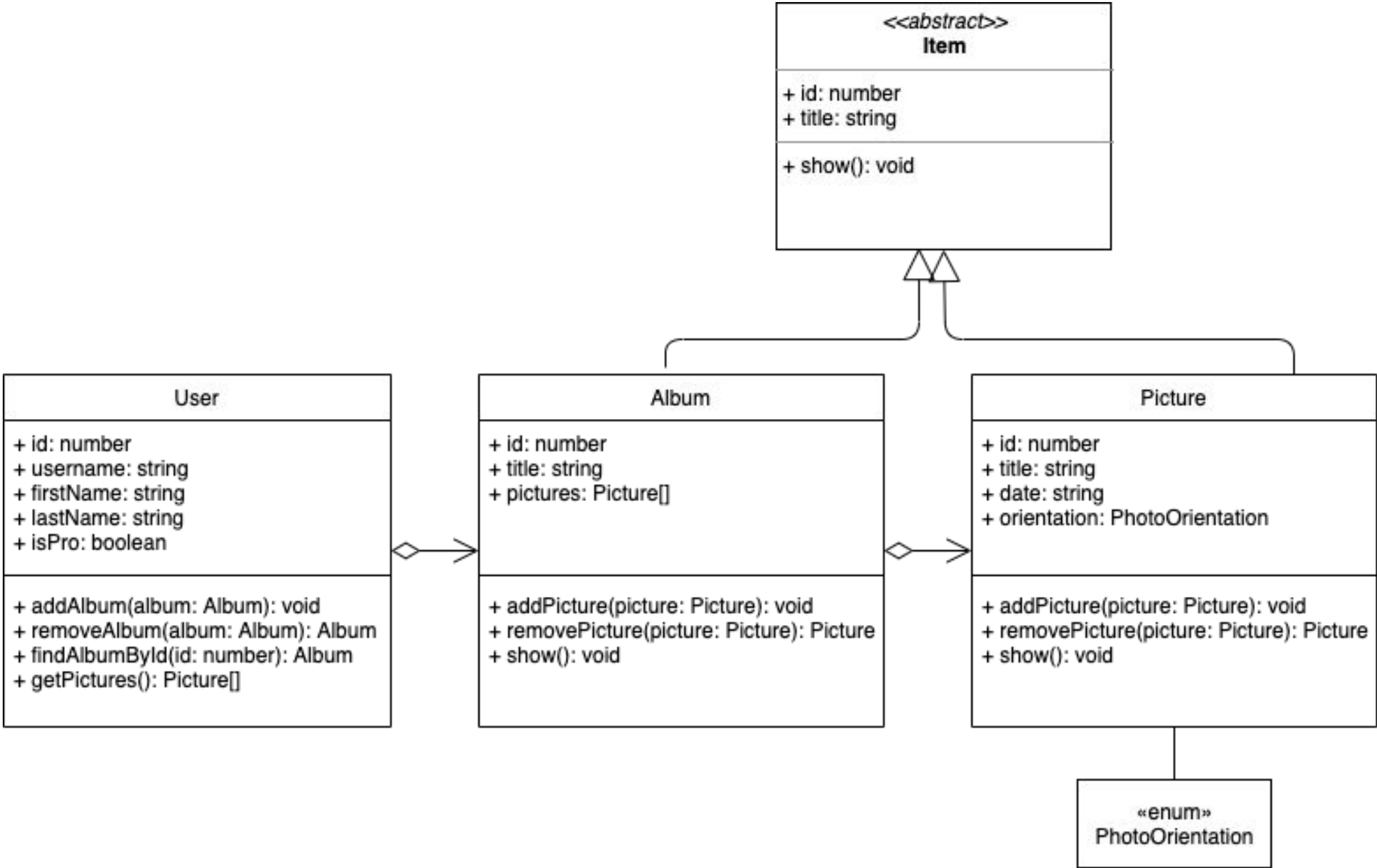
Propiedades Estáticas y Propiedades de Solo Lectura

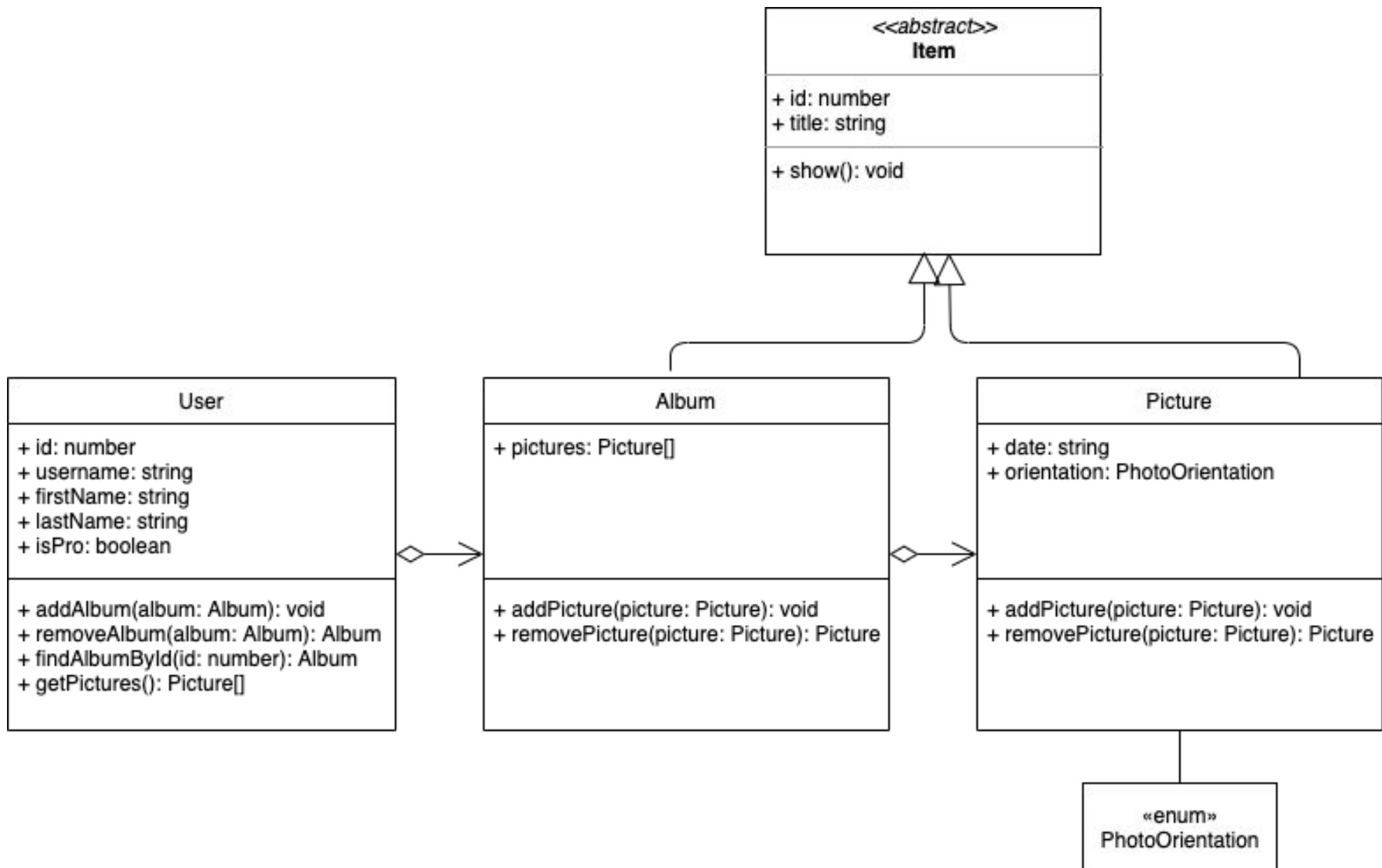
Las clases por lo general definen atributos y métodos aplicables a las instancias de las mismas.

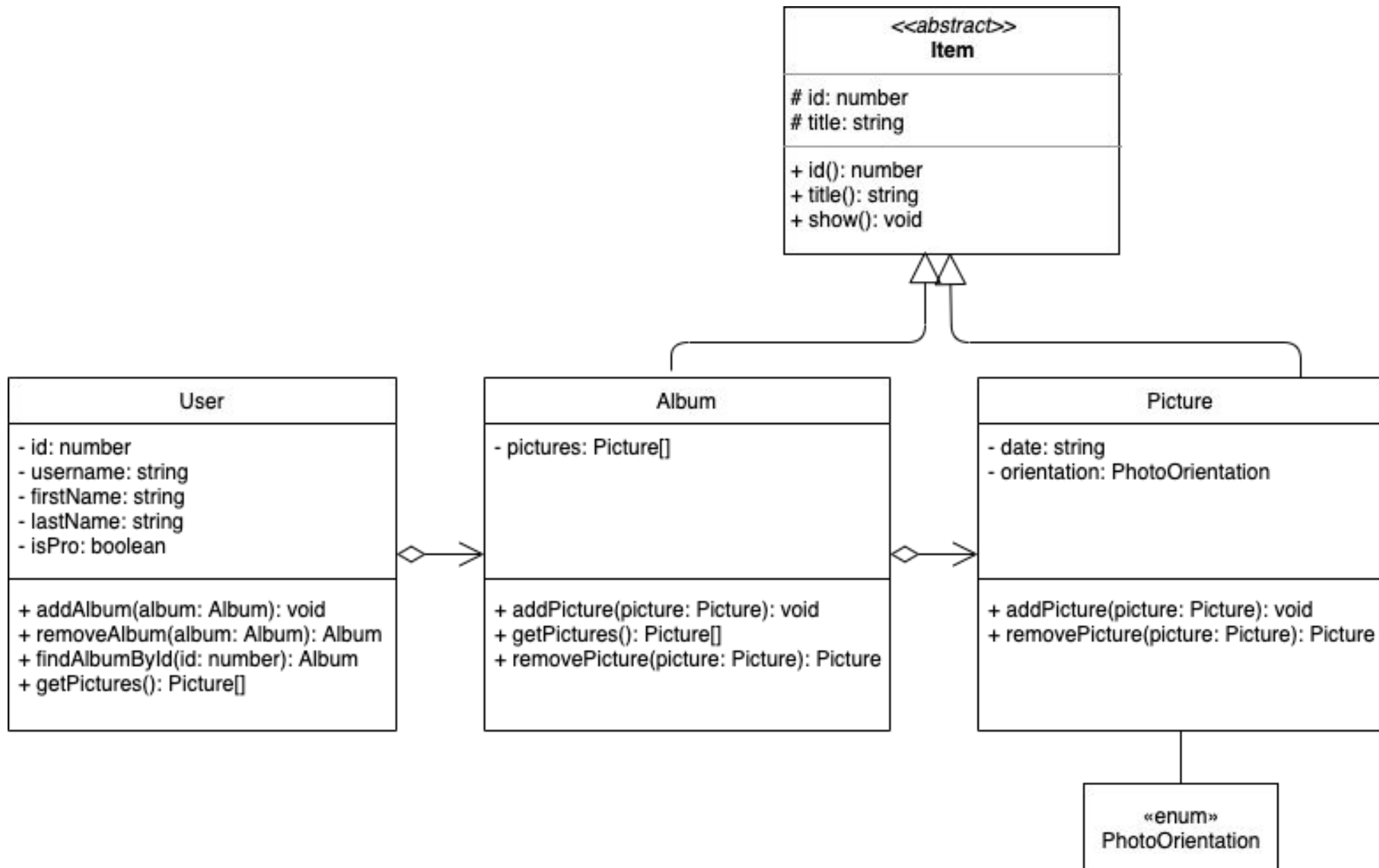
A través de la palabra reservada `static` se puede definir un miembro visible a nivel de clase.

Propiedades Estáticas y Propiedades de Solo Lectura

Al igual que las Interfaces, podemos usar la palabra reservada `readonly` para marcar el miembro de una clase como solo lectura









Resumen

Módulos en TypeScript



Módulos en TypeScript

Los módulos en TypeScript proveen un mecanismo para una mejor organización del código y promueven su reutilización

A partir de ECMAScript 2015, los módulos son parte nativa del lenguaje JavaScript



The background of the entire image is a dense, chaotic pile of blue LEGO bricks of various shapes and sizes, including 1x1 squares, 1x2 rectangles, and 2x2 squares. The bricks are scattered across the frame, creating a textured, three-dimensional effect. Overlaid on this background are five white-outlined, rounded rectangular boxes, each containing a software engineering principle in Spanish. The boxes are arranged in a loose, non-linear pattern: 'Mantenible' and 'Reusable' are at the top; 'Testeable' and 'Ordenado' are in the middle; and 'Encapsulado' is at the bottom center.

Mantenible

Reusable

Testeable

Ordenado

Encapsulado

Importando y Exportando en Módulos

Importando y Exportando en Módulos

Generalmente se define un módulo con la idea de agrupar código relacionado.

Podemos tomar criterios en torno a la funcionalidad, *features*, utilitarios, modelos, etc

Importando y Exportando en Módulos

Los miembros de módulo interactúan con el uso de las palabras reservadas `import` y `export`

Principio de Responsabilidad Única

Idealmente, un archivo debería tener un propósito o responsabilidad única: definir una clase, una interfaz, un enumerado, etc

Esto mejora la legibilidad de código, facilita su lectura, testing y favorece su mantenimiento



Resolviendo Módulos



Resolviendo Módulos

TypeScript resuelve la ubicación de módulos observando referencias relativas y no relativas.

Posteriormente intenta localizar el módulo usando una **estrategia de resolución de módulos**.

Estrategia de Resolución de Módulos



```
$ tsc --moduleResolution node
```

```
$ tsc --moduleResolution classic
```

Estrategia de Resolución de Módulos

classic	node
Módulos AMD, System, ES2015	Módulos CommonJS o UMD
Poco configurable	Más opciones de configuración

"module": "<valor>"(tsconfig.json)

Classic: Import Relativo

```
// Archivo: /typescript/photo-app/main.ts  
import { Picture } from './picture;
```

```
/typescript/photo-app/picture.ts  
/typescript/photo-app/picture.d.ts
```

Classic: Import No-Relativo

```
// Archivo: /typescript/photo-app/main.ts  
import { Picture } from 'picture';
```

```
/typescript/photo-app/picture.ts  
/typescript/photo-app/picture.d.ts
```

```
/typescript/picture.ts  
/typescript/picture.d.ts  
(continúa buscando en el árbol de directorios)
```

Node: Import Relativo

```
// Archivo: /typescript/photo-app/main.ts  
import { Picture } from './picture;
```

```
/typescript/photo-app/picture.ts  
/typescript/photo-app/picture.tsx  
/typescript/photo-app/picture.d.ts
```


Node: Import Relativo

```
// Archivo: /typescript/photo-app/main.ts  
import { Picture } from './picture;
```

```
/typescript/photo-app/picture.ts  
/typescript/photo-app/picture.tsx  
/typescript/photo-app/picture.d.ts
```

```
/typescript/photo-app/picture/package.json ("typings")
```

Node: Import Relativo

```
// Archivo: /typescript/photo-app/main.ts  
import { Picture } from './picture;
```

```
/typescript/photo-app/picture.ts  
/typescript/photo-app/picture.tsx  
/typescript/photo-app/picture.d.ts
```

```
/typescript/photo-app/picture/package.json ("typings")
```

```
/typescript/photo-app/index.ts  
/typescript/photo-app/index.tsx  
/typescript/photo-app/index.d.ts
```

Node: Import No-Relativo

```
// Archivo: /typescript/photo-app/main.ts  
import { Picture } from 'picture';
```

```
/typescript/photo-app/node_modules/picture.ts  
/typescript/photo-app/node_modules/picture.tsx  
/typescript/photo-app/node_modules/picture.d.ts  
/typescript/photo-app/node_modules/picture/package.json  
/typescript/photo-app/node_modules/index.ts
```

Node: Import No-Relativo

```
// Archivo: /typescript/photo-app/main.ts  
import { Picture } from 'picture';
```

```
/typescript/photo-app/node_modules/picture.ts  
/typescript/photo-app/node_modules/picture.tsx  
/typescript/photo-app/node_modules/picture.d.ts  
/typescript/photo-app/node_modules/picture/package.json  
/typescript/photo-app/node_modules/index.ts
```

```
/typescript/node_modules/picture.ts  
/typescript/node_modules/picture.tsx  
/typescript/node_modules/picture.d.ts  
/typescript/node_modules/index.ts  
/typescript/node_modules/index.tsx  
/typescript/node_modules/index.d.ts
```

(continua buscando en el árbol de directorios)

Webpack y Agrupación de Módulos

Agregando el archivo package.json



```
$ npm init -y
```

Instalación de TypeScript y Webpack

```
$ npm install typescript webpack webpack-cli  
--save-dev
```

webpack.config.js

```
module.exports = {  
  mode: 'production',  
  entry: './main.ts',  
  devtool: 'inline-source-map',  
  resolve: {  
    extensions: ['.ts', '.js'],  
  },  
  output: {  
    filename: 'bundle.js',  
  },  
}
```


Instalación de ts-loader

```
$ npm install ts-loader --save-dev
```

webpack.config.json

```
module.exports = {  
  mode: 'production',  
  entry: './src/main.ts',  
  devtool: 'inline-source-map',  
  module: {  
    rules: [  
      {  
        test: /\.ts$/,  
        use: 'ts-loader',  
        exclude: /node_modules/  
      }  
    ]  
  },  
  resolve: {  
    extensions: ['.ts', '.js'],  
  },  
  output: {  
    filename: 'bundle.js',  
  }  
}
```

```
// Estructura de archivos
```

```
| - ts-project/  
  | - src/  
    | - main.ts  
    | - photo-app/  
      | - index.ts  
  | - dist/  
    index.html  
    bundle.js  
  | - package.json  
  | - tsconfig.json  
  | - webpack.config.js
```



Cierre del Curso