PREGRADO

Unidad 1: Overview

TYPESCRIPT LANGUAJE BASICS

SI705 | Arquitectura de Aplicaciones Web

Al finalizar la unidad de aprendizaje, el estudiante describe el proceso de software realizado aplicando el paradigma orientado a objetos combinado con aspectos de la programación funcional, utilizando el lenguaje Java y frameworks de actualidad, para desarrollar aplicaciones web básicas en un ambiente de desarrollo colaborativo.

TypeScript (TS)

- Es un lenguaje de programación construido a un nivel superior de JavaScript (JS).
- Fue creado por Microsoft en 2012 y, desde entonces, su adopción no ha hecho más que crecer.
- Google decidió adoptarlo como lenguaje por defecto para desarrollar con Angular.
- Hoy en día, podemos desarrollar con TypeScript en cualquiera de los frameworks, como son React para el frontend o Node para el backend.

AGENDA

BASIC TYPES

NON-CONCRETE TYPES

INTERFACES AND

CLASSES

ASSERTION AND UNION

TYPES

GENERICS

Para definir una variable de un tipo simplemente debemos poner el nombre de la variable después el signo de dos puntos (:) y el tipo de la variable.

```
const NombreVariable : string = 'A';
```

Basic Types

```
let isDone: boolean = false;  //Boolean


let decimal: number = 6;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;
let big: bigint = 100n;

let color: string = "blue"; //String
color = "red";

let fullName: string = `Bob Bobbington`;
let age: number = 37;
let sentence: string = `Hello, my name is ${fullName}.

I'll be ${age + 1} years old next month.`;

let sentence: string =
  "Hello, my name is " + fullName + ".\n\n" + "I'll be " + (age + 1) +
  " years old next month.";
```

Array

```
let list: number[] = [1, 2, 3];

let list: Array<number> = [1, 2, 3];
```

Tuple

```
// Declare a tuple type
let x: [string, number];

// Initialize it
x = ["hello", 10]; // OK

// Initialize it incorrectly x = [10, "hello"]; // Error

Type 'number' is not assignable to type 'string'.
Type 'string' is not assignable to type 'number'.

// OK
console.log(x[0].substring(1));
console.log(x[1].substring(1)); // Error

Property 'substring' does not exist on type 'number'.

x[3] = "world";

Tuple type '[string, number]' of length '2' has no element at index '3’.

console.log(x[5].toString());

Object is possibly 'undefined'.
Tuple type '[string, number]' of length '2' has no element at index '5'.
```

Enum

```
enum Color {
  Red,
  Green,
  Blue,
}
let c: Color = Color.Green;
```

```
enum Color {
  Red = 1,
  Green,
  Blue,
}
let c: Color = Color.Green;
```

```
enum Color {
  Red = 1,
  Green = 2,
  Blue = 4,
}
let c: Color = Color.Green;
```

```
enum Color {
  Red = 1,
  Green,
  Blue,
}
let colorName: string = Color[2];


// Displays 'Green'
console.log(colorName);
```

AGENDA

BASIC TYPES

NON-CONCRETE TYPES

INTERFACES AND

CLASSES

ASSERTION AND UNION

TYPES

GENERICS

Unknown

```
let notSure: unknown = 4;
notSure = "maybe a string instead";

// OK, definitely a boolean
notSure = false;

declare const maybe: unknown;
// 'maybe' could be a string, object, boolean, undefined, or other types
const aNumber: number = maybe;
Type 'unknown' is not assignable to type 'number'.

if (maybe === true) {
  // TypeScript knows that maybe is a boolean now
  const aBoolean: boolean = maybe;
  // So, it cannot be a string
  const aString: string = maybe;
Type 'boolean' is not assignable to type 'string'.
}

if (typeof maybe === "string") {
  // TypeScript knows that maybe is a string
  const aString: string = maybe;
  // So, it cannot be a boolean
  const aBoolean: boolean = maybe;
Type 'string' is not assignable to type 'boolean'.
}
```

Any

```
declare function getValue(key: string): any;
// OK, return value of 'getValue' is not checked
const str: string = getValue("myString");


let looselyTyped: any = 4;
// OK, ifItExists might exist at runtime
looselyTyped.ifItExists();
// OK, toFixed exists (but the compiler doesn't check)
looselyTyped.toFixed();


let strictlyTyped: unknown = 4;
strictlyTyped.toFixed();
Object is of type 'unknown'.


let looselyTyped: any = {};
let d = looselyTyped.a.b.c.d;
//  ^ = let d: any
```

Void, Null, Undefined, Never

```
function warnUser(): void {
  console.log("This is my warning message");
}

let unusable: void = undefined;
// OK if `--strictNullChecks` is not given
unusable = null;

// Not much else we can assign to these variables!
let u: undefined = undefined;
let n: null = null;

// Function returning never must not have a reachable end point
function error(message: string): never {
  throw new Error(message);
}

// Inferred return type is never
function fail() {
  return error("Something failed");
}

// Function returning never must not have a reachable end point
function infiniteLoop(): never {
  while (true) {}
}
```

AGENDA

BASIC TYPES

NON-CONCRETE TYPES

INTERFACES AND

CLASSES

UNION TYPES

ASSERTION AND

GENERICS

Interfaces

En TypeScript, una interfaz solo contiene la definición de métodos y propiedades, no su implementación. Es la funcionalidad de la clase que realiza la conexión entre la interfaz proporcionando la conexión con todos los parámetros de la interfaz.

```
interface PersonData {
name: string;
weight? : number;
age: number;
hairColor? : string;
height: number;
}
```

Interfaces funcionales

```
interface LabeledValue {
  label: string;
}

function printLabel(labeledObj: LabeledValue) {
  console.log(labeledObj.label);
}

let myObj = { size: 10, label: "Size 10 Object" };
printLabel(myObj);
```

Interfaces funcionales

Optional properties.

```
interface SquareConfig {
  color?: string;
  width?: number;
}

function createSquare(config: SquareConfig): { color: string; area: number } {
  let newSquare = { color: "white", area: 100 };
  if (config.color) {
    newSquare.color = config.color;
  }
  if (config.width) {
    newSquare.area = config.width * config.width;
  }
  return newSquare;
}

let mySquare = createSquare({ color: "black" });
```

Interfaces

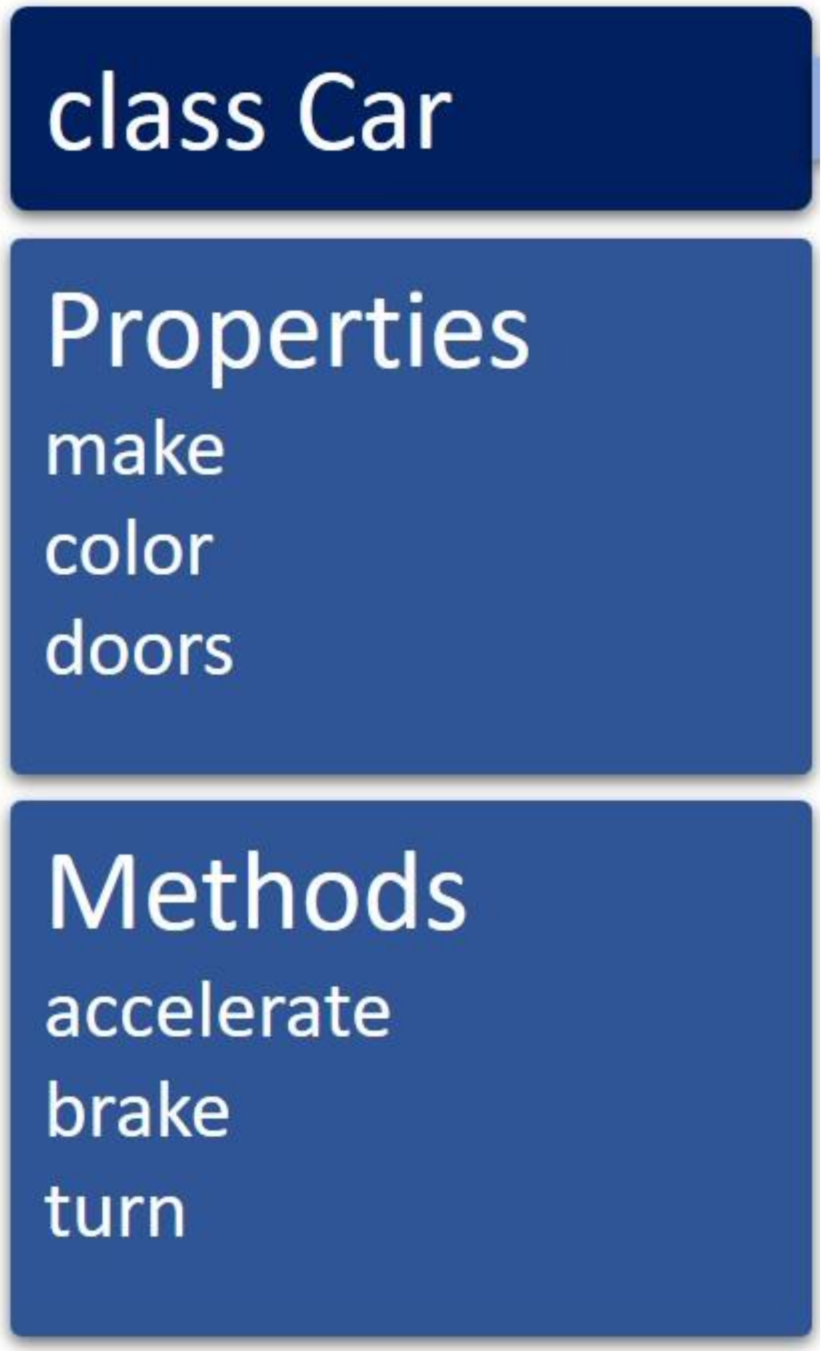
Readonly properties.

```
interface Point {
  readonly x: number;
  readonly y: number;
}

let p1: Point = { x: 10, y: 20 };
p1.x = 5; // error!
```

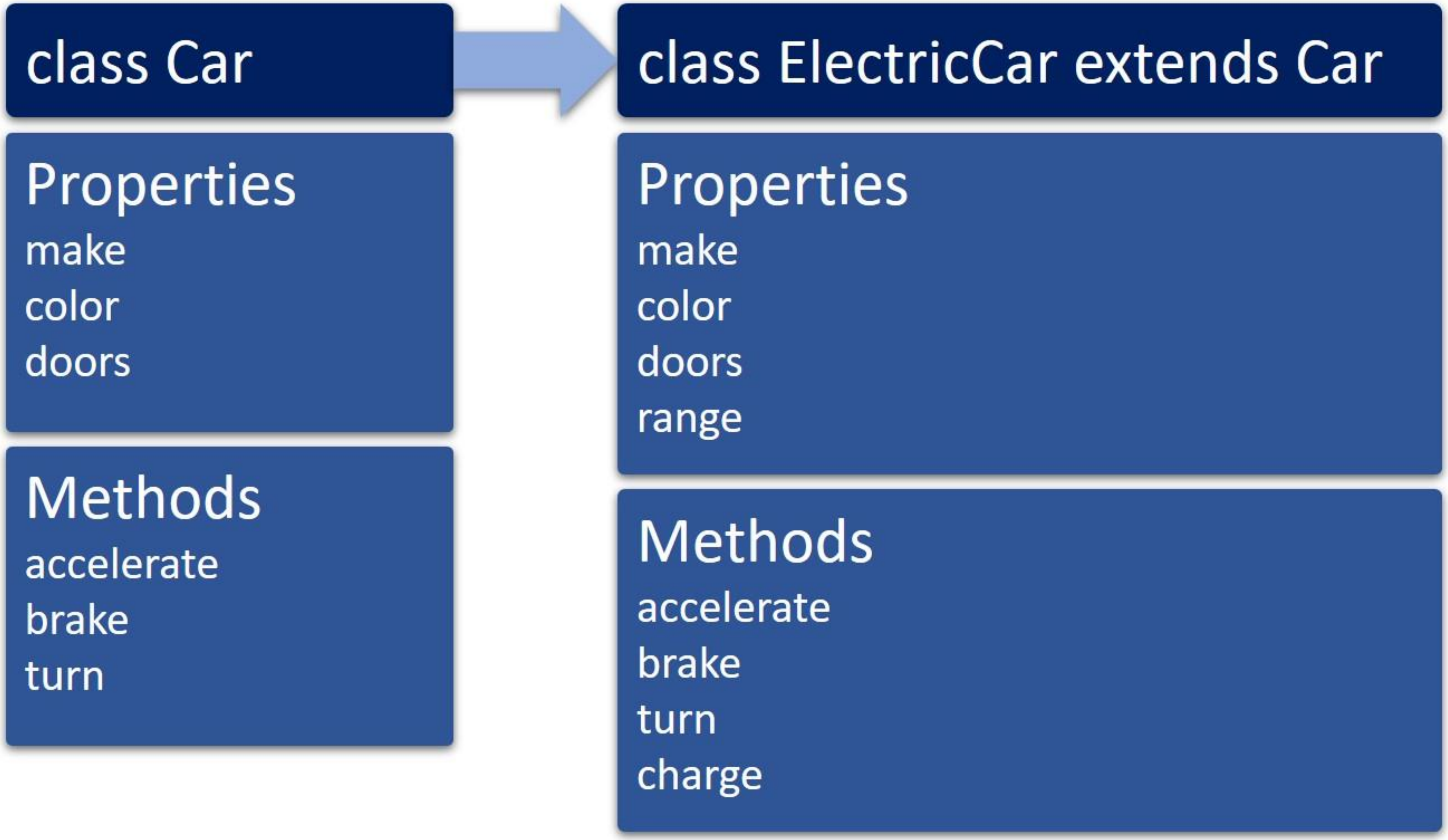
Classes

En TypeScript, las clases son otra forma de definir la forma de un objeto, además de describir tipos de objetos con interfaces y funciones.



Classes

Inheritance



Classes

Inheritance

```
class Animal {
  name: string;
  constructor(theName: string) {
    this.name = theName;
  }
  move(distanceInMeters: number = 0) {
    console.log(`${this.name} moved 
${distanceInMeters}m.`);
  }
}

class Snake extends Animal {
  constructor(name: string) {
    super(name);
  }
  move(distanceInMeters = 5) {
    console.log("Slithering...");
    super.move(distanceInMeters);
  }
}
```

```
class Horse extends Animal {
  constructor(name: string) {
    super(name);
  }
  move(distanceInMeters = 45) {
    console.log("Galloping...");
    super.move(distanceInMeters);
  }
}

let sam = new Snake("Sammy the 
Python");
let tom: Animal = new Horse("Tommy 
the Palomino");

sam.move();
tom.move(34);
```

AGENDA

BASIC TYPES

NON-CONCRETE TYPES

INTERFACES AND

CLASSES

UNION TYPES

ASSERTION AND

GENERICS

Union

Union con Fields que sean comunes.

```
interface Bird {
  fly(): void;
  layEggs(): void;
}

interface Fish {
  swim(): void;
  layEggs(): void;
}

declare function getSmallPet(): Fish | Bird;

let pet = getSmallPet();
pet.layEggs();


// Only available in one of the two possible types
pet.swim();
Property 'swim' does not exist on type 'Bird | Fish'.
Property 'swim' does not exist on type 'Bird'.
```

Union

Discriminating Unions

```
type NetworkLoadingState = {
  state: "loading";
};

type NetworkFailedState = {
  state: "failed";
  code: number;
};

type NetworkSuccessState = {
  state: "success";
  response: {
    title: string;
    duration: number;
    summary: string;
  };
};

// Create a type which represents only one of the above types
// but you aren't sure which it is yet.
type NetworkState =
  | NetworkLoadingState
  | NetworkFailedState
  | NetworkSuccessState;
```

Union

Discriminating Unions

```
function logger(state: NetworkState): string {
  // Right now TypeScript does not know which of the three
  // potential types state could be.

  // Trying to access a property which isn't shared
  // across all types will raise an error
  state.code;

  Property 'code' does not exist on type 'NetworkState'.
  Property 'code' does not exist on type 'NetworkLoadingState’.

  // By switching on state, TypeScript can narrow the union
  // down in code flow analysis
  switch (state.state) {
    case "loading":
      return "Downloading...";
    case "failed":
      // The type must be NetworkFailedState here,
      // so accessing the `code` field is safe
      return `Error ${state.code} downloading`;
    case "success":
      return `Downloaded ${state.response.title} - ${state.response.summary}`;
  }
}
```

AGENDA

BASIC TYPES

NON-CONCRETE TYPES

INTERFACES AND

CLASSES

UNION TYPES

ASSERTION AND

GENERICS

Type Assertions

Los type assertions son una forma de decirle al compilador que debe confiar en ti, porque entiendes lo que estas haciendo.
Un type assertion es como el type cast de otros lenguajes, pero se ejecuta sin verificar ninguna verificación o reestructurando datos.
No tiene impacto durante la ejecución y es manejado exclusivamente por el compilador

```
let someValue: unknown = "this is a string";

let strLength: number = (someValue as string).length;

let someValue: unknown = "this is a string";

let strLength: number = (<string>someValue).length;
```

Generic Types

Generic Classes

El conjunto de recursos disponibles con genéricos aumenta cuando los usamos con clases. Así podemos tener una clase en la que declaremos el uso de un tipo genérico.

```
class GenericNumber<T> {
  zeroValue: T;
  add: (x: T, y: T) => T;
}


let myGenericNumber = new
GenericNumber<number>();
myGenericNumber.zeroValue = 0;
myGenericNumber.add = function(x, y) {
  return x + y;
};
```

Generic Types

Generic Constraints

```
function loggingIdentity<T>(arg: T): T {
  console.log(arg.length); // Error: Property 'length' does not exist on type 'T'.
  return arg;
}


// We create an interface that describes out constraint.
interface Lengthwise {
  length: number;
}

function loggingIdentity<T extends Lengthwise>(arg: T): T {
  console.log(arg.length); // Now we know it has a .length property, so no more error
  return arg;
}
```

REFERENCIAS

Para profundizar

https://www.typescriptlang.org/docs/handbook/intro.html

https://www.typescriptlang.org/

https://hackr.io/blog/top-10-web-development-frameworks-in-2020

https://angular.io/

https://angular.io/guide/architecture

https://angular.io/tutorial

https://angular.io/start

PREGRADO

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC
Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco
Lima 33 - Perú
T 511 313 3333
https://www.upc.edu.pe

exígete, innova

LAUREATE
INTERNATIONAL
UNIVERSITIES