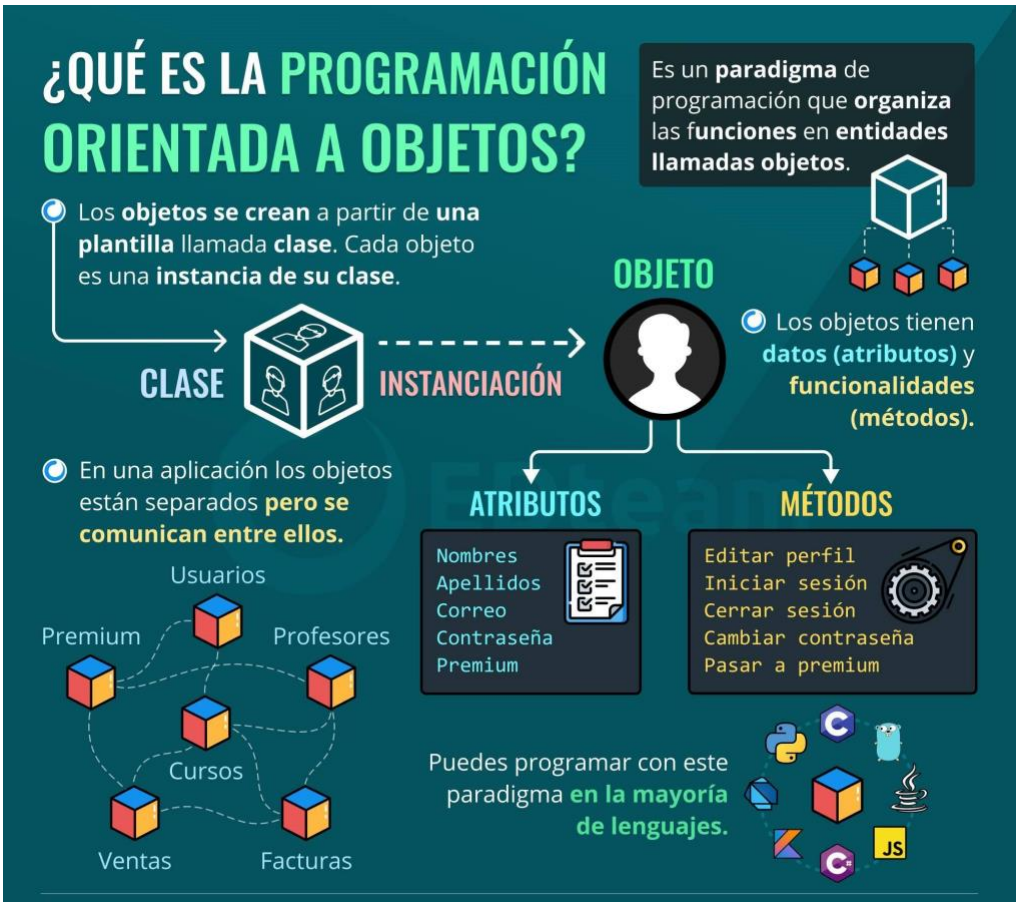
DART 101

Objetivos

- Definición programación orientada a objetos
- Clases en Dart
    - Propiedades
    - Métodos
    - Constructores
    - Seters – Geters
- Herencia
- Enumeraciones

Programación orientada a objetos



DART 101

Clases en Dart

Dart es un lenguaje orientado a objetos con clases y herencia basada en mixin. Cada objeto es una instancia de una clase y todas las clases excepto Null descienden de Object.
Las ventajas de las clases en Dart es que tenemos un mejor control de datos en la aplicación y se evitan posibles errores que pueden suceder por ejemplo al usar Mapas o Listas.
A continuacion se muestra la estructura basica de una clase en Dart:

Propiedades

Son valores de variables que puede tener una clase, desde que se implementó el null safety, se debe especificar el valor al incializar (null o no null). Todas las variables de instancia generan un método *getter* implícito . Las variables de instancia no finales y las variables de instancia sin inicializadores también generan un método *de establecimiento*late final implícito.
Si no se establece un valor en el incializador el editor muestra un error ya que la variable debe ser incializada o especificada como null.

```dart
class Animal {

  /// Atributos o propiedades
  String nombre;
  int edad;
  String color;

  /// Constructor

  /// Getters y Setters

  /// Metodos

}
```

Valor especificado como null

```dart
class Animal {

  /// Atributos o propiedades
  String? nombre;
  int? edad;
  String? color;

  /// Constructor

  /// Getters y Setters

  /// Metodos

}
```

Variables con incialización

```dart
class Animal {

  /// Atributos o propiedades
  String nombre = "Sin nombre";
  int edad = 0;
  String color = "Sin color";

  /// Constructor

  /// Getters y Setters

  /// Metodos

}
```

No es muy recomendable especificar en las propiedades con un valor incial o como null, ya que perdería la razón de usar una clase para crear varios objetos, pero para objetos prácticos se usa de esta manera.

Solamente con incializar la clase ya podemos acceder a sus valores.

```
Run | Debug
void main(List<String> args) {
    final animalDefault = Animal();
    print(animalDefault.nombre); // Sin nombre
    print(animalDefault.edad); // 0
    print(animalDefault.color); // Sin color
}
```

Se puede editar los valores internos de la clase de la siguiente manera

```
Run | Debug
void main(List<String> args) {
    final perro = Animal();
    perro.nombre = "Firulais";
    perro.edad = 10;
    perro.color = "Cafe";
    print(perro.nombre); // Firulais
    print(perro.edad); // 10
    print(perro.color); // Cafe
}
```

Importante: se puede establecer las propiedades como privadas, la función de esto es tener un control sobre valores no mutables dentro de la clase y mutar estos valores dentro de los setters y getters respectivamente. Las propiedades privadas se establecen con _, esto no solo sirve para propiedades, tambien puede ser usado en métodos, clases, etc.

```
String _sonido = "Sin sonido";
```

Constructores

Como se comentó en el apartado anterior, no es recomendable establecer valores iniciales en las propiedades, para que el editor no muestre el error se utiliza el constructor.

Un constructor recibe parametros, estos pueden ser posicionales o nombrados.

A continuacion, se muestra una clase con un constructor con parametros posicionales.

```dart
class Animal {
  /// Atributos o propiedades
  String nombre;
  int edad;
  String color;

  /// Constructor
  Animal(this.nombre, this.edad, this.color);

  /// Getters y Setters

  /// Metodos
  @override
  String toString() {
    return "Nombre: $nombre, Edad: $edad, Color: $color";
  }
}
```

Cuando se maneja clases se recomienda usar parametros nombrados cuando el número de propiedades supera la unidad, esto para que futuros desarrolladores que toquen este código tengan una mejor facilidad de lectura.

```dart
// Constructor con parametros nombrados
Animal({required this.nombre, required this.edad, required this.color});
```

Nota: los parámetros del constructor siempre deben ser obligatorios, a menos que se especifique lo contrario.

```dart
Animal({
  this.nombre = "Sin nombre",
  this.edad = 0,
  this.color = "Sin color",
});
```

Ver mas sobre constructores: https://dart.dev/language/constructors

Principio de inmutabilidad

Hasta ahora se han detallado ejemplos en donde las propiedades de la clase son mutables, pero al desarrollar en flutter se recomienda que en lo que mas se pueda usar propiedades inmutables, a menos que la circunstancia lo amerite, para que las propiedades sean inmutables se debe especificar la palabra reservada final en las propiedades.

```dart
class Animal {
  /// Atributos o propiedades
  final String nombre;
  final int edad;
  final String color;
```

El uso del final evita que se pueda actualizar el valor de la propiedad cuando se está haciendo el uso de la clase.

¿Cómo puedo editar esa propiedad?

Para editar la propiedad se recomienda el uso de un método “copyWith” que crea otra una copia de la clase en otra instancia de memoria, con el fin de mantener inmutada a la clase principal.

Getters

Get es una palabra reservada que permite obtener un valor simple o un valor tratado de dentro de una clase.

```dart
// getter
String get nombreMayusculas {
  return this.nombre.toUpperCase();
}
// lambda
String get nombreMayusculasLambda => this.nombre.toUpperCase();
```

Setters

La palabra reservada set se usa cuando se quiere modificar el valor de una propiedad de una clase, siempre y cuando esta sea mutable.

```dart
// setter
set nombreMayusculas(String nombre) {
  this.nombre = nombre.toUpperCase();
}

// lambda
set nombreMayusculasLambda(String nombre) =>
    this.nombre = nombre.toUpperCase();
```

¿Cuál es la diferencia entre modificar u obtener un valor mediante getters/setter y acceder directamente a la clase?

Realmente no existe ninguna diferencia entre usar el uno o el otro, pero en ciertos momentos se va a usar esa clase en varios lugares de una aplicación por lo que se recomienda centralizar esos métodos dentro de la clase para que el mantenimiento del código sea más facil.

Métodos

Un método es una función, por lo tanto se puede establecer cualquier cantidad de métodos a una clase. La diferencia con el setter es que un método no necesariamente muta el valor de una propiedad.

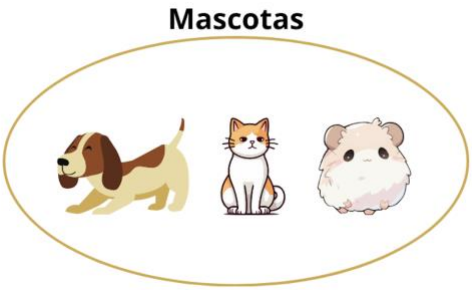
```dart
/// Metodos

// método copyWith para crear una copia de las propiedades de un objeto
Animal copyWith({
  String? nombre,
  int? edad,
  String? color,
}) {
  return Animal(
    nombre: nombre ?? this.nombre,
    edad: edad ?? this.edad,
    color: color ?? this.color,
  );
}
```

Ver mas sobre los métodos: https://dart.dev/language/methods

Herencia

Dart tiene una herencia unica, no podemos tener una herencia múltiple.

Para conocer que es la herencia hay que imaginarse tres mascotas; un perro, un gato y un hamster.



Perfectamente se puede definir una clase para cada uno de las mascotas, como se muestra en la siguiente imagen:

```dart
class Perro {
  final nombre;
  final edad;

  Perro(this.nombre, this.edad);
}

class Gato {
  final nombre;
  final edad;

  Gato(this.nombre, this.edad);
}

class Hamster {
  final nombre;
  final edad;

  Hamster(this.nombre, this.edad);
}
```

Se puede notar que las 3 clases comparten dos propiedades similares entonces se puede englobar perfectamente todo a una clase Mascota:

```dart
class Mascota {
  final nombre;
  final edad;

  Mascota(this.nombre, this.edad);
}
```

Se puede usar simplemente la clase mascota, per perdería sentido si en un futuro se necesita agregar propiedades o métodos propios que correspondan a un perro, a un gato o a un hamster, a continuación se muestra lo que no sería correcto:

```dart
class Mascota {
  final nombre;
  final edad;

  final String colorHamster;
  final String colorGato;
  final String colorPerro;

  Mascota(this.nombre, this.edad, this.colorHamster, this.colorGato, this.colorPerro);
}
```

Para evitar el problema de propiedades que no corresponden a un objeto en concreto se utiliza **extends**, para definir la clase Mascota como la clase padre y la clase Perro, Gato y Hamster como clases hijos, que van a heredar todas las propiedades del padre pero tienen la facilidad de ocupar nuevas propiedades o métodos que solo les correspondan a ellas.

```
class Perro extends Mascota {
  final color;

  Perro(this.color, nombre, edad) : super(nombre, edad);
  void ladrar() {
    print("Guau!");
  }
}

class Gato extends Mascota {
  final color;
  Gato(this.color, nombre, edad) : super(nombre, edad);
  void maullar() {
    print("Miau!");
  }
}
```

¿Qué es la palabra super?

La palabra reservada **super** en el constructor permite el paso de variables hasta la clase padre antes de su contruccion, de esa manera se asegura que el padre va a tener las propiedades que su hijo heredó. Cuando la palabra **super** se usa dentro de la clase, esta hace referencia a las propiedades, metodos, setters, getters del padre.

Para conocer mas sobre herencia recomiendo estos artículos:

https://carloszr.com/herencia-en-dart-extends-en-dart-herencia-de-clases/

https://josuecruzdigital.com/diferencias-entre-extends-implements-y-with-en-dart/

https://gist.github.com/rsalhuana/af9a3e268e6becdf8b1d359dbe0d482a

https://dart.dev/language/extend

https://dart.dev/language/mixins

DART 101

Enumeraciones

Se usa para representar un valor fijo.

```dart
// Enum simple
enum DiaSemana { Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo }

// Enum descriptivo
enum DiaSemanaDescriptivo {
  Lunes("Hoy es Lunes"),
  Martes("Hoy es Martes"),
  Miercoles("Hoy es Miercoles"),
  Jueves("Hoy es Jueves"),
  Viernes("Hoy es Viernes"),
  Sabado("Hoy es Sabado"),
  Domingo("Hoy es Domingo");

  final String descripcion;
  const DiaSemanaDescriptivo(this.descripcion);
}
```