DART – Class (Clases)

Leave a Comment / Dart / By jbachi

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 descienden de Object. La herencia basada en Mixin significa que, aunque cada clase (excepto Object) tiene exactamente una superclase, un cuerpo de clase se puede reutilizar en varias jerarquías de clases. Los métodos de extensión son una forma de agregar funcionalidad a una clase sin cambiar la clase o crear una subclase.

Índice [ocultar]

Estructura de una clase en DART

```
class Persona {

  // Propiedades

  // Get y sets

  // Constructores

  // metodos

}
```

Muy bien, ya sabemos como cual es la estructura base de una clase aun no todas estas parte resultas obligatorioas. Vamos a crear nuestra primera clase:

Mi Primera Clase

```
main() {

  // Defino una variable del tipo de la clase en este caso Persona
  final persona = Persona();

  //Asigno valor a las propiedades de manera directa
  persona.nombre = 'Joe';
  persona.edad = 25;

  //Acceso al metodo sobreescrito
  print(persona.toString());

}

// Las clases se definen con la palbra Class
// Para el nombre de la misma de coloca la primera letra de cada palabra en mayuscula: LaPrimeraEnMayuscula
class Persona {

  // Propiedades
  String nombre ;
  int edad;

  // Get y sets

  // Constructores

  // metodos

  // Con override indicamos que sobreescribimos el metodo "padre"
  @override
  String toString() {
    return 'Nombre: ${nombre}, Edad: ${edad}';
  }

}
```

Propiedades Privadas

Son propiedades que solo pueden ser usadas dentro de la clases (de forma interna), la misma se logra colocando UNDERSCORE antes del nombre de la variables.

OJO: todas las propiedades de una clase por defecto son publica, asi que para ellas no hay que hacer ningún ajuste

```
main() {

  // Defino una variable del tipo de la clase en este caso Persona
  final persona = Persona();

  //Asigno valor a las propiedades de manera directa
  persona.nombre = 'Joe';
  persona.edad = 25;
  // persona._valorPrivado // No podras acceder porque es Privado

  //Acceso al metodo sobreescrito
  print(persona.toString());

}

// Las clases se definen con la palbra Class
// Para el nombre de la misma de coloca la primera letra de cada palabra en mayuscula: LaPrimeraEnMayuscula
class Persona {

  // Propiedades
  String nombre ;
  int edad;
  int _valorPrivado;

  // Get y sets

  // Constructores

  // metodos

  // Con override indicamos que sobreescribimos el metodo "padre"
  @override
  String toString() {
    return 'Nombre: ${nombre}, Edad: ${edad}';
  }

}
```

Setters y Getters

Métodos que sirven para simular una propiedad. Permiten definir y obtener los datos de las variables privadas. A tener en cuenta:

- Se suele colocar el mismo nombre de la propiedad que se quieres trabajar pero sin el UNDERSCORE (pero no es obligatorio puede poner el nombre que desees)
- Las palabras claves SET y GET contiene definiciones implícitas, como por ejemplo el SET automáticamente retorna una función de tipo VOID.
- En el SET debes indicar el tipo de dato a Retornar (lo puedes dejar dinámico pero no es recomendable).
- En ambos casos puedes ejecutar más lineas de código antes de finalizar, como ejemplo, en el GET he multiplicado el valor por 50 (esta es la utilidad de aplicar estos métodos, no multiplicar por 50 jejeje sino poder trabajar el dato al enviarlo o recibirlo)

```
main() {

  // Defino una variable del tipo de la clase en este caso Persona
  final persona = Persona();

  //Asigno valor a las propiedades de manera directa
  persona.nombre = 'Joe';
  persona.edad = 25;
  // persona._valorPrivado // No podras acceder porque es Privado
  persona.valorPrivado = 50;

  //Acceso al metodo sobreescrito
  print(persona.toString());

}

// Las clases se definen con la palabra Class
// Para el nombre de la misma de coloca la primera letra de cada palabra en mayuscula: LaPrimeraEnMayuscula
class Persona {

  // Propiedades
  String nombre ;
  int edad;
  int _valorPrivado;

  // Get y sets
  int get valorPrivado{
    //return _valorPrivado * 10;
    return _valorPrivado;
  }

  set valorPrivado(int valor){
    _valorPrivado = valor;
  }

  // Constructores

  // metodos

  // Con override indicamos que sobreescribimos el metodo "padre"
  @override
  String toString() {
    return 'Nombre: ${nombre}, Edad: ${edad}, Valor Privado: ${_valorPrivado}';
  }

}
```

Constructores

Son los métodos que se crean al momento de crear una instancia de la clase. En palabras mas simple, el método principal por defecto.

- Por defecto sin se crear ningún constructor no pasa nada, DART puede funcionar sin definirlo.
- El constructor por defecto que se ejecuta al implementar debe tener el mismo nombre de la clase.
- También existen los Constructores por nombre que veremos mas adelante.

Constructor básico:

```
// Constructores
Persona(String nombre, int edad){
  this.nombre = nombre;
  this.edad = edad;
}
```

Constructor Resumido

```
Persona(this.nombre, this.edad);
```

Constructor con Argumentos por nombre:

```
Persona({this.nombre, this.edad});
```

Constructor con valor por Defecto:

```
Persona({this.nombre, this.edad = 18});
```

Puedes aprender más sobre los argumentos presiona aquí

Ahora como estamos solicitando datos en el constructor principal la forma de hacer la instancia cambiar a algo similar a esto (para cada ejemplo varia un poco, pero entendiendo lo de los parámetros estarás bien):

```
final persona = Persona(nombre: 'Joe Bachi',edad: 21);
```

Constructores con Nombre

Permite tener mas de un constructor, los cuales se ejecutaran dependiendo de como se instacia la clase. Cuando usa este tipo de constructor, es importarte aclarar, que solo se ejecuta ese y no el constructor por defecto. Aqui un ejemplo de como se veria una clase con tres constructores:

```
// Constructores
Persona({this.nombre, this.edad = 15});

Persona.mayorDeEdad(this.nombre){
  this.edad = 18;
}

Persona.adulta(this.nombre){
  this.edad = 45;
}
```

y los llamaríamos de la siguiente forma:

```
final persona = Persona(nombre: 'Joe Bachi',edad: 21);
final personaMayor = Persona.mayorDeEdad('Evelin');
final persona2 = Persona.adulta('Luis');
```

Propiedades Finales

Estas propiedades se pueden asignar mas no se podrán editar. Resulta útil cuando necesitamos asegurarnos que no se modificaran ciertos valores una vez se haga la instancia de la clase.

```
main() {

  final a = Figura(10);
  a.lado // No Permitido
  a.area // No Permitido

}

class Figura {
  final int lado;
  final int area;

  Figura(int lado):
    this.lado = lado,
    this.area = lado * lado;
}
```

Fíjese que todo parece estar en orden hasta el momento de crear el constructor, ya que usamos el símbolo de dos punto ":", este nos permite usar los parámetros del mismo antes de que se inicialice la clase, ya que de lo contrario los valores de LADO y AREA nunca se les asignaría un valor y diera un error ya que las variables FINAL deben tener un valor al inicializarse.

A su vez, los valores de LADO y AREA ya no podrán ser modificados para esa instancia.

← Previous Post Next Post →

Leave a Comment

Your email address will not be published. Required fields are marked *

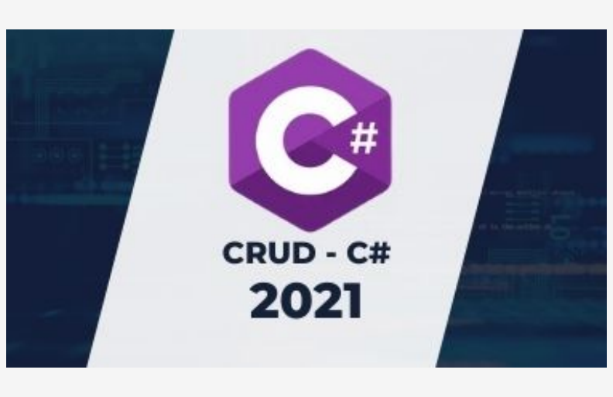
Type here..

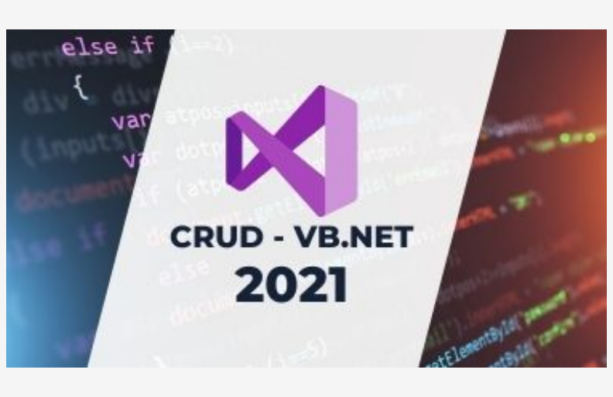
Name*
Email*
Website

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment »

Curso Gratis





Ver todos los cursos

Categorías

- .NET (1)
  - VB .NET (1)
- Dart (6)
- Flutter (1)
- Herramientas (1)
- Laravel (3)
- Programación (4)

Reciente

- Instrucción y Primeros Paso con Laragon
- MessageBox a Profundidad VB .NET
- DART – Class (Clases)
- DART – String to int – int to String (Conversiones)
- DART – Future – Async – Await