Clase: Una clase es una plantilla o molde para crear objetos. Define las propiedades (atributos) y comportamientos (métodos) que todos los objetos creados a partir de ella compartirán.

Objeto: Un objeto es una instancia de una clase. Cada objeto tiene un estado (los valores de sus atributos) y un comportamiento (las acciones que puede realizar a través de sus métodos).

Abstracción: La abstracción es el proceso de identificar las características esenciales y relevantes de un objeto y omitir las características no esenciales. La abstracción permite simplificar el modelo mental de un sistema y enfocarse en las partes importantes.

```dart
class Empleado {
  late int id;
  late String nombre;

  void mensajeHorario() {
    print('El empleado $nombre ha cumplido su horario');
  }
}
```

```dart
import 'Empleado.dart';

Run | Debug
main() {
  Empleado empleado = Empleado();
  empleado.id = 1;
  empleado.nombre = 'Juan';
  empleado.mensajeHorario();
}
```

```
PROBLEMS 5  OUTPUT  TERMINAL  DEBUG CONSOLE
El empleado Juan ha cumplido su horario
```

Constructor por defecto: No requiere de ningún parámetro para inicializar un objeto, ya existe cuando se crea una clase y se define creando un método con el mismo nombre de la clase.

```dart
class Empleado {
  late int id;
  late String nombre;

  Empleado() {
    print('Constructor por defecto de la clase Empleado');
  }

  void mensajeHorario() {
    print('El empleado $nombre ha cumplido su horario');
  }
}
```

```dart
import 'Empleado.dart';

Run | Debug
main() {
  Empleado empleado = Empleado();
  empleado.id = 1;
  empleado.nombre = 'Juan';
  empleado.mensajeHorario();
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
Constructor por defecto de la clase Empleado
El empleado Juan ha cumplido su horario
Exited
```

Constructor con parámetros: Es un constructor que recibe uno o más parámetros para inicializar los atributos de un objeto. Un constructor con parámetros se utiliza cuando se requiere inicializar los atributos de un objeto con valores específicos que se pasan como argumentos. No puede existir en la misma clase un constructor por defecto y uno con parámetros.

```dart
class Empleado {
  late int id;
  late String nombre;

  Empleado(id, nombre) {
    this.id = id;
    this.nombre = nombre;
  }

  void mensajeHorario() {
    print('El empleado $nombre ha cumplido su horario');
  }
}
```

```dart
import 'Empleado.dart';

Run | Debug
main() {
  Empleado empleado = Empleado(1, 'Juan');
  empleado.mensajeHorario();
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
El empleado Juan ha cumplido su horario
```

Constructores nombrados: Se puede definir constructores con un nombre que uno defina y estos pueden existir múltiples constructores.

```dart
class Empleado {
  late int id;
  late String nombre;
  late bool jornadaConcluida;

  Empleado(id, nombre) {
    this.id = id;
    this.nombre = nombre;
  }
  Empleado.jornadaConcluida(this.id, this.nombre, this.jornadaConcluida) {}

  bool cumpleHorario() {
    return jornadaConcluida;
  }

  void mensajeHorario() {
    print('El empleado $nombre ha cumplido su horario');
  }
}
```

```dart
import 'Empleado.dart';

Run | Debug
main() {
  Empleado empleado = Empleado(1, 'Juan');
  print(empleado.id);
  print(empleado.nombre);
  empleado.mensajeHorario();

  Empleado empleado2 = Empleado.jornadaConcluida(2, 'Pedro', true);
  print(empleado2.id);
  print(empleado2.nombre);
  print(empleado2.cumpleHorario());
  empleado2.mensajeHorario();
}
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE   Filter (e.g. text, !exclu
1
Juan
El empleado Juan ha cumplido su horario
2
Pedro
true
El empleado Pedro ha cumplido su horario
```

Encapsulamiento: El encapsulamiento es el principio de POO que establece que los datos y el comportamiento de un objeto deben ser ocultados y protegidos de la manipulación externa. Los objetos sólo pueden interactuar con otros objetos a través de una interfaz definida (los métodos públicos de la clase).

```dart
class Empleado {
  int id;
  String nombre;

  late int _diasLaborados; // Propiedad privada

  Empleado(this.id, this.nombre);

  //Setter de dias laborados
  void set setDiasTrabajados(int dias) => _diasLaborados = dias;

  //Getter de días laborados
  int get getDiasLaborados => _diasLaborados;
}
```

```dart
import 'Empleado.dart';

Run | Debug
main() {
  Empleado empleado = Empleado(1, 'Juan');

  empleado.setDiasTrabajados = 5;

  print(empleado.getDiasLaborados);
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
5
```

Herencia: La herencia es un mecanismo mediante el cual una clase puede heredar propiedades y comportamientos de otra clase. La clase que se hereda se llama "clase base" o "superclase", y la clase que hereda se llama "clase derivada" o "subclase". La herencia permite reutilizar el código existente y crear jerarquías de clases.

Clase abstracta: Una clase abstracta es una clase que no puede ser instanciada directamente, sino que sirve como plantilla para definir otras clases.

```dart
abstract class Empleado {
  late int _id;
  late String _nombre;
  late double _salario;

  void set setID(int id) => _id = id;
  int get getID => _id;

  void set setNombre(String nombre) => _nombre = nombre;
  String get getNombre => _nombre;

  void set setSalario(double salario) => _salario = salario;
  double get getSalario => _salario;

  String imprimirPantalla() {
    return 'El trabajador ${_nombre}, con el ID ${_id}. \n'
        'Tiene un salario de ${_salario * 5.5} pesos por semana. \n';
  }
}
```

```dart
import 'Empleado.dart';

class Vendedor extends Empleado {
  late String _idCliente;

  void set setIdCliente(String idCliente) => _idCliente = idCliente;
  String get getIdCliente => _idCliente;
}
```

```dart
import 'Empleado.dart';

class Chofer extends Empleado {
  late String _vehiculoAsignado;

  void set setVehiculo(String vehiculo) => _vehiculoAsignado = vehiculo;
  String get getVehiculo => _vehiculoAsignado;
}
```

```dart
import 'Chofer.dart';
import 'Vendedor.dart';

Run | Debug
main() {
  Chofer chofer = Chofer();
  chofer.setNombre = 'Juan';
  chofer.setID = 1;
  chofer.setSalario = 1000;
  chofer.setVehiculo = 'Toyota';

  Vendedor vendedor = Vendedor();
  vendedor.setNombre = 'Pedro';
  vendedor.setID = 2;
  vendedor.setSalario = 2000;
  vendedor.setIdCliente = 'C-001';

  print(chofer.imprimirPantalla());
  print(vendedor.imprimirPantalla());
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
El trabajador Juan, con el ID 1.
Tiene un salario de 5500.0 pesos por semana.

El trabajador Pedro, con el ID 2.
Tiene un salario de 11000.0 pesos por semana.
```

Polimorfismo: El polimorfismo es la capacidad de los objetos de diferentes clases para responder a los mismos mensajes (llamadas a métodos) de diferentes maneras. Esto se logra mediante la implementación de interfaces comunes o mediante la redefinición de métodos en clases derivadas.

```dart
import 'Empleado.dart';

class Chofer extends Empleado {
  late String _vehiculoAsignado;

  void set setVehiculo(String vehiculo) => _vehiculoAsignado = vehiculo;
  String get getVehiculo => _vehiculoAsignado;

  @override
  String imprimirPantalla() {
    return super.imprimirPantalla() +
        'Vehículo asignado: ${_vehiculoAsignado}. \n';
  }
}
```

```dart
import 'Empleado.dart';

class Vendedor extends Empleado {
  late String _idCliente;

  void set setIdCliente(String idCliente) => _idCliente = idCliente;
  String get getIdCliente => _idCliente;

  @override
  String imprimirPantalla() {
    return super.imprimirPantalla() +
        'Tipo de clientes asignados: ${_idCliente}. \n';
  }
}
```

```dart

  print(chofer.imprimirPantalla());
  print(vendedor.imprimirPantalla());
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
El trabajador Juan, con el ID 1.
Tiene un salario de 5500.0 pesos por semana.
Vehículo asignado: Toyota.

El trabajador Pedro, con el ID 2.
Tiene un salario de 11000.0 pesos por semana.
Tipo de clientes asignados: C-001.
```

Interfaz: Define una serie de operaciones que deben ser realizadas por un objeto, sin especificar cómo se llevan a cabo. Dart no tiene una sintaxis especial para interfaces, una clase normal puede funcionar como una interfaz, a esto se le conoce como interfaces implícitas.

```dart
class Empleado {
  var id;
  var nombre;
  var salario;

  void calcularSalario() {}
}
```

```dart
class Conducta {
  void estadoConducta() {}
}
```

```dart
import 'Conducta.dart';
import 'Empleado.dart';

class Vendedor implements Empleado, Conducta {
  @override
  var id;

  @override
  var nombre;

  @override
  var salario;

  @override
  void calcularSalario() {
    print('El salario del empleado ${nombre} es ${salario * 6}');
  }

  @override
  void estadoConducta() {
    print('La conducta del empleado ${nombre} fué ejemplar');
  }
}
```

```dart
import 'Conducta.dart';
import 'Empleado.dart';

class Chofer implements Empleado, Conducta {
  @override
  var id;

  @override
  var nombre;

  @override
  var salario;

  @override
  void calcularSalario() {
    print('El salario del empleado ${nombre} es ${salario * 6}');
  }

  @override
  void estadoConducta() {
    print('La conducta del empleado ${nombre} fué incorrecta');
  }
}
```

```dart
import 'Chofer.dart';
import 'Vendedor.dart';

Run | Debug
main() {
  Chofer chofer = Chofer();
  chofer.id = 1;
  chofer.nombre = 'Juan';
  chofer.salario = 1000;
  chofer.calcularSalario();
  chofer.estadoConducta();

  Vendedor vendedor = Vendedor();
  vendedor.id = 2;
  vendedor.nombre = 'Pedro';
  vendedor.salario = 1000;
  vendedor.calcularSalario();
  vendedor.estadoConducta();
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
El salario del empleado Juan es 6000
La conducta del empleado Juan fué incorrecta
El salario del empleado Pedro es 6000
La conducta del empleado Pedro fué ejemplar
```