Dart Programming - Classes

Dart is an object-oriented language. It supports object-oriented programming features like classes, interfaces, etc. A **class** in terms of OOP is a blueprint for creating objects. A **class** encapsulates data for the object. Dart gives built-in support for this concept called **class**.

Declaring a Class

Use the **class** keyword to declare a **class** in Dart. A class definition starts with the keyword class followed by the **class name**; and the class body enclosed by a pair of curly braces. The syntax for the same is given below −

Syntax

```
class class_name {
   <fields>
   <getters/setters>
   <constructors>
   <functions>
}
```

The **class** keyword is followed by the class name. The rules for identifiers must be considered while naming a class.

A class definition can include the following −

- **Fields** − A field is any variable declared in a class. Fields represent data pertaining to objects.
- **Setters and Getters** − Allows the program to initialize and retrieve the values of the fields of a class. A default getter/ setter is associated with every class. However, the default ones can be overridden by explicitly defining a setter/ getter.
- **Constructors** − responsible for allocating memory for the objects of the class.
- **Functions** − Functions represent actions an object can take. They are also at times referred to as methods.

These components put together are termed as the **data members** of the class.

Example: Declaring a class

```
class Car {
   // field
   String engine = "E1001";

   // function
   void disp() {
      print(engine);
   }
}
```

The example declares a class **Car**. The class has a field named **engine**. The **disp()** is a simple function that prints the value of the field **engine**.

Creating Instance of the class

To create an instance of the class, use the **new** keyword followed by the class name. The syntax for the same is given below −

Syntax

```
var object_name = new class_name([ arguments ])
```

- The **new** keyword is responsible for instantiation.
- The right-hand side of the expression invokes the constructor. The constructor should be passed values if it is parameterized.

Example: Instantiating a class

```
var obj = new Car("Engine 1")
```

Accessing Attributes and Functions

A class's attributes and functions can be accessed through the object. Use the '.' dot notation (called as the **period**) to access the data members of a class.

```
//accessing an attribute
obj.field_name

//accessing a function
obj.function_name()
```

Example

Take a look at the following example to understand how to access attributes and functions in Dart −

Live Demo

```
void main() {
   Car c= new Car();
   c.disp();
}
class Car {
   // field
   String engine = "E1001";

   // function
   void disp() {
      print(engine);
   }
}
```

The **output** of the above code is as follows −

```
E1001
```

Dart Constructors

A constructor is a special function of the class that is responsible for initializing the variables of the class. Dart defines a constructor with the same name as that of the class. A constructor is a function and hence can be parameterized. However, unlike a function, constructors cannot have a return type. If you don't declare a constructor, a default **no-argument constructor** is provided for you.

Syntax

```
Class_name(parameter_list) {
   //constructor body
}
```

Example

The following example shows how to use constructors in Dart −

Live Demo

```
void main() {
   Car c = new Car('E1001');
}
class Car {
   Car(String engine) {
      print(engine);
   }
}
```

It should produce the following **output** −

```
E1001
```

Named Constructors

Dart provides **named constructors** to enable a class define **multiple constructors**. The syntax of named constructors is as given below −

Syntax : Defining the constructor

```
Class_name.constructor_name(param_list)
```

Example

The following example shows how you can use named constructors in Dart −

Live Demo

```
void main() {
   Car c1 = new Car.namedConst('E1001');
   Car c2 = new Car();
}
class Car {
   Car() {
      print("Non-parameterized constructor invoked");
   }
   Car.namedConst(String engine) {
      print("The engine is : ${engine}");
   }
}
```

It should produce the following **output** −

```
The engine is : E1001
Non-parameterized constructor invoked
```

The this Keyword

The **this** keyword refers to the current instance of the class. Here, the parameter name and the name of the class's field are the same. Hence to avoid ambiguity, the class's field is prefixed with the **this** keyword. The following example explains the same −

Example

The following example explains how to use the **this** keyword in Dart −

Live Demo

```
void main() {
   Car c1 = new Car('E1001');
}
class Car {
   String engine;
   Car(String engine) {
      this.engine = engine;
      print("The engine is : ${engine}");
   }
}
```

It should produce the following **output** −

```
The engine is : E1001
```

Dart Class – Getters and Setters

Getters and **Setters**, also called as **accessors** and **mutators**, allow the program to initialize and retrieve the values of class fields respectively. Getters or accessors are defined using the **get** keyword. Setters or mutators are defined using the **set** keyword.

A default getter/setter is associated with every class. However, the default ones can be overridden by explicitly defining a setter/ getter. A getter has no parameters and returns a value, and the setter has one parameter and does not return a value.

Syntax: Defining a getter

```
Return_type  get identifier
{
}
```

Syntax: Defining a setter

```
set identifier
{
}
```

Example

The following example shows how you can use **getters** and **setters** in a Dart class −

Live Demo

```
class Student {
   String name;
   int age;

   String get stud_name {
```

```
      return name;
   }

   void set stud_name(String name) {
      this.name = name;
   }

   void set stud_age(int age) {
      if(age<= 0) {
        print("Age should be greater than 5");
      }  else {
         this.age = age;
      }
   }

   int get stud_age {
      return age;
   }
}
void main() {
   Student s1 = new Student();
   s1.stud_name = 'MARK';
   s1.stud_age = 0;
   print(s1.stud_name);
   print(s1.stud_age);
}
```

This program code should produce the following **output** −

```
Age should be greater than 5
MARK
Null
```

Class Inheritance

Dart supports the concept of Inheritance which is the ability of a program to create new classes from an existing class. The class that is extended to create newer classes is called the parent class/super class. The newly created classes are called the child/sub classes.

A class inherits from another class using the 'extends' keyword. **Child classes inherit all properties and methods except constructors from the parent class**.

Syntax

```
class child_class_name extends parent_class_name
```

Note − Dart doesn't support multiple inheritance.

Example: Class Inheritance

In the following example, we are declaring a class **Shape**. The class is extended by the **Circle** class. Since there is an inheritance relationship between the classes, the child class, i.e., the class **Car** gets an implicit access to its parent class data member.

Live Demo

```
void main() {
   var obj = new Circle();
   obj.cal_area();
}
```

```
class Shape {
   void cal_area() {
      print("calling calc area defined in the Shape class");
   }
}
class Circle extends Shape {}
```

It should produce the following **output** −

```
calling calc area defined in the Shape class
```

Types of Inheritance

Inheritance can be of the following three types −

- **Single** − Every class can at the most extend from one parent class.
- **Multiple** − A class can inherit from multiple classes. Dart doesn't support multiple inheritance.
- **Multi-level** − A class can inherit from another child class.

Example

The following example shows how multi-level inheritance works −

Live Demo

```
void main() {
   var obj = new Leaf();
   obj.str = "hello";
   print(obj.str);
}
class Root {
   String str;
}
class Child extends Root {}
class Leaf extends Child {}
//indirectly inherits from Root by virtue of inheritance
```

The class **Leaf** derives the attributes from Root and Child classes by virtue of multi-level inheritance. Its **output** is as follows −

```
hello
```

Dart – Class Inheritance and Method Overriding

Method Overriding is a mechanism by which the child class redefines a method in its parent class. The following example illustrates the same −

Example

Live Demo

```
void main() {
   Child c = new Child();
   c.m1(12);
}
class Parent {
   void m1(int a){ print("value of a ${a}");}
}
class Child extends Parent {
```

```
   @override
   void m1(int b) {
      print("value of b ${b}");
   }
}
```

It should produce the following **output** −

```
value of b 12
```

The number and type of the function parameters must match while overriding the method. In case of a mismatch in the number of parameters or their data type, the Dart compiler throws an error. The following illustration explains the same −

Live Demo

```
import 'dart:io';
void main() {
   Child c = new Child();
   c.m1(12);
}
class Parent {
   void m1(int a){ print("value of a ${a}");}
}
class Child extends Parent {
   @override
   void m1(String b) {
      print("value of b ${b}");
   }
}
```

It should produce the following **output** −

```
value of b 12
```

The static Keyword

The **static** keyword can be applied to the data members of a class, i.e., **fields** and **methods**. A static variable retains its values till the program finishes execution. Static members are referenced by the class name.

Example

Live Demo

```
class StaticMem {
   static int num;
   static disp() {
      print("The value of num is ${StaticMem.num}")  ;
   }
}
void main() {
   StaticMem.num = 12;
   // initialize the static variable }
   StaticMem.disp();
   // invoke the static method
}
```

It should produce the following **output** −

```
The value of num is 12
```

The super Keyword

The **super** keyword is used to refer to the immediate parent of a class. The keyword can be used to refer to the super class version of a **variable, property,** or **method**. The following example illustrates the same −

Example

Live Demo

```
void main() {
   Child c = new Child();
   c.m1(12);
}
class Parent {
   String msg = "message variable from the parent class";
   void m1(int a){ print("value of a ${a}");}
}
class Child extends Parent {
   @override
   void m1(int b) {
      print("value of b ${b}");
      super.m1(13);
      print("${super.msg}")   ;
   }
}
```

It should produce the following **output** −

```
value of b 12
value of a 13
message variable from the parent class
```