Dart Object-Oriented Concepts

Dart is an object-oriented programming language, and it supports all the concepts of object-oriented programming such as classes, object, inheritance, and abstract classes. As the name suggests, it focuses on the object and objects are the real-life entities. The Object-oriented programming approach is used to implement the concept like polymorphism, data-hiding, etc. The main goal of oops is to reduce programming complexity and do several tasks simultaneously.

Class

Dart classes are defined as the blueprint of the associated objects. A Class is a user-defined data type that describes the characteristics and behavior of it. To get all properties of the class, we must create an object of that class.

```
class ClassName {
  <fields>
 <getter/setter>
  <constructor>
  <functions>
}
```

Object

An object is a real-life entity such as a table, human, car, etc. The object has two characteristics - state and behavior. Let's take an example of a car which has a name, model name, price and behavior moving, stopping, etc. The object-oriented programming offers to identify the state and behavior of the object.

We can access the class properties by creating an object of that class. In Dart, The object can be created by using a new keyword followed by class name. The syntax is given below.

```
var objectName = new ClassName(<constructor_arguments>)
```

Defining a Class in Dart

Dart provides **class** keyword followed by a class name is used to define a class; all fields and functions are enclosed by the pair of curly braces ({}).

Note - According to the naming convention rule of identifiers, the first letter of the class name must be capital and use no separators.

```
// Defining class
class Student {
  var stdName;
  var stdAge;
  var stdNo;
  // Class Function
  showStdInfo() {
    print("Student Name is : ${stdName}");
    print("Student Age is : ${stdAge}");
    print("Student Roll Number is : ${stdNo}");
  }
}
```

Dart Object

Dart is object-oriented programming, and everything is treated as an object in Dart. An object is a variable or instance of the class used to access the class's properties. Objects have two features - state and behavior. Suppose a man is an object with a state (name, age, health) and behavior (walking, running, and sleeping). Programming objects are theoretically similar to the real-life objects; they also have state and behavior. An object is created from a template which is known as class.

The fields of the classes are stored as object states, whereas the method represents an object's behavior.

Creating Class Objects in Dart

After creating the class, we can create an instance or object of that class which we want to access its fields and functions. The **new** keyword is used to declare class followed by the class name. The general syntax of creating an object of a class is given below.

```
var object_name  = new class_name(<constructor_arguments>);
```

```dart
class Student {
  var stdName;
  var stdAge;
  var stdNo;
  // defining class function
  showStdInfo() {
    print("Student Name is : $stdName");
    print("Student Age is : $stdAge");
    print("Student Roll Number is : $stdNo");
  }
}
void main() {
  // Creating object called std
  var std = new Student();
  std.stdName = “Ali";
  std.stdAge = 24;
  std.stdNo = 2190001;
// Accessing class Function
  std.showStdInfo();
}
```

```
Student Name is : Ali
Student Age is : 24
Student Roll Number is : 2190001
```

constructor

A constructor is a different type of function which is created with same name as its class name. The constructor is used to initialize an object when it is created. When we create the object of class, then constructor is automatically called. It is quite similar to class function, but it has no explicit return type. The generative constructor is the most general form of the constructor, which is used to create a new instance of a class.

It is option to declare within the class. All class have own constructor but if we don't declare or forget then Dart compiler will create default constructor automatically by passing the default value to the member variable. If we declare our own constructor, then default constructor will be ignored.

Creating Constructor in Dart

A constructor has the same name as its class name, and it doesn't return any value. Suppose if we have class Student then the constructor's name should be also **Student**.

```
class ClassName {
   ClassName() {
}
}
```

```
void main() {
  // Creating an object
  Student std = new Student("Jones", 26);
}

class Student {
  // Declaring a construstor
  Student(String str, int age) {
    print("The name is: ${str}");
    print("The age is: ${age}");
  }
}
```

```
The name is: Jones
The age is: 26
```

Types of Constructors

There are three types of constructors in Dart.

❑ Default Constructor or no-arg Constructor

A Constructor which has no parameter is called default constructor or no-arg constructor. It is automatically created (with no argument) by Dart compiler if we don't declare in the class. The Dart compiler ignores the default constructor if we create a constructor with argument or no argument.

❑ Parameterized Constructor

We can also pass the parameters to a constructor that type of constructor is called parameterized constructor. It is used to initialize instance variables. Sometimes, we need a constructor which accepts single or multiple parameters. The parameterized constructors are mainly used to initialize instance variable with own values.

❑ Named Constructors

The named constructors are used to declare the multiple constructors in single class. The syntax is given below.

```
className.constructor_name(param_list)
```

```dart
void main() {
  // Creating an object
  Student std1 = new Student(); // object for Default constructor
  Student std2 = new Student.namedConst("Computer Science"); // named const.
}
class Student {
  // Declaring a constructor
  Student() {
    print("The example of the named constructor");
  }
  // Second constructor
  Student.namedConst(String branch) {
    print("The branch is: ${branch}");
  }
}
```

```
The example of the named constructor
The branch is: Computer Science
```

Dart this Keyword

The **this** keyword is used to refer the current class object. It indicates the current instance of the class, methods, or constructor. It can be also used to call the current class methods or constructors. It eliminates the uncertainty between class attributes and the parameter names when are the same. If we declare the class attributes same as the parameter name, that situation will create ambiguity in the program, then the this keyword can remove the ambiguity by prefixing the class attributes. It can be passed as an argument in the class method or constructors.

```
class Mobile {
  String? modelname;
  int? man_year;
  // Creating constructor
  Mobile(modelname, man_year) {
    modelname = modelname;
    man_year = 2020;
    print("model name is: $modelname,
manufacture year is: $man_year");  }}
void main() {
  Mobile mob = new Mobile("iPhone 11 ", 2020);
}
```

```
class Mobile {
  String? modelname;
  int? man_year;
  // Creating constructor
  Mobile(modelname, man_year) {
    this.modelname = modelname;
    this.man_year = 2020;
    print("model name is:
$modelname, manufacture year is:
$man_year"); }}
void main() {
  Mobile mob = new Mobile("IPhone
11", 2020);}
```

Local Variable

Local variables are defined in the methods, constructors, or blocks. It is created when we create a method or constructor, and it has scope only inside them. We cannot use a local variable outside the method, constructor, or block.

Class Variable

Class variable is also known as the **static member** variable, which is used to declare using the static keyword. It is declared in the class, but outside a constructor, method or a block. All instances share one copy of the class variable or we can say that class variables are common to all instances of that class.

Instance Variable

Instance variable is also known as the non-static, variable which is used to declare without the static keyword. The instance variables are specific by an object. These variables can be accessed using the instance of that class.

Dart Static Variable

A variable which is declared using the static keyword inside the class is called Dart static keyword. These are the member of the class instead of a specific instance. The static variables are treated the same for all instances of the class; it means a single copy of the static variable is shared among all instances of classes. It allocates memory once and at the class loading and uses throughout the program.

Point to Remember -

- The static variable is also identified as a class variable.
- Single copy of the static variable is shared among the instance of a class.
- It can be accessed using the class name. We don't need to create an object of that class they belong to.
- The static variables can be accessed directly in the static methods.

Declaring Static Variable

Dart provides the static keyword to declare the static variable. It is declared by using the static keyword followed by the variable name. The syntax is given below.

```
static [data_type] [variable_name];
```

Accessing Static Variable

We can access the static variable by using the class name itself instead of creating an object of it.

Dart Static Method

The concept of the static method is also similar to static variable. The static methods are the member of the class instead of the class instance. The static methods can use only static variables and can invoke the static method of the class. We don't need to create an instance of class to access it. The static method is useful when we want to use it in other classes.

Points to Remember

- The static methods are the member class instead of its object.
- Static methods are also identifying as class methods.
- We can access static methods using the class name.
- A particular copy of the static method is distributed among all the instances of a class.

Declaring Static Methods

We can declare the static method by using the static keyword followed by the method name with the return type. The syntax is given below.

```
static return_type method_name() {
 //statement(s)  }
```

Calling Static Method

The static methods can be called by using the class name, which they belong to instead of creating an object.

```dart
class Student {
  static String? stdBranch; // Declaring static variable
  String? stdName;
  int? stdNo;
  showStdInfo() {
    print("Student's name is: $stdName");
    print("Student's salary is: $stdNo");
    print("Student's branch name is: $stdBranch");
  }
}
void main() {
  Student std1 = new Student();
  Student std2 = new Student();
  // Assigning value of static variable using class name
  Student.stdBranch = "Computer Science";
  std1.stdName = “Ahmed";
  std1.stdNo = 2190013;
  std1.showStdInfo();
  std2.stdName = “Omer";
  std2.stdNo = 2190014;
  std2.showStdInfo(); }
```

```
Student's name is: Ahmed
Student's salary is: 2190013
Student's branch name is:
Computer Science
Student's name is: Omer
Student's salary is: 2190014
Student's branch name is:
Computer Science
```

Dart Inheritance

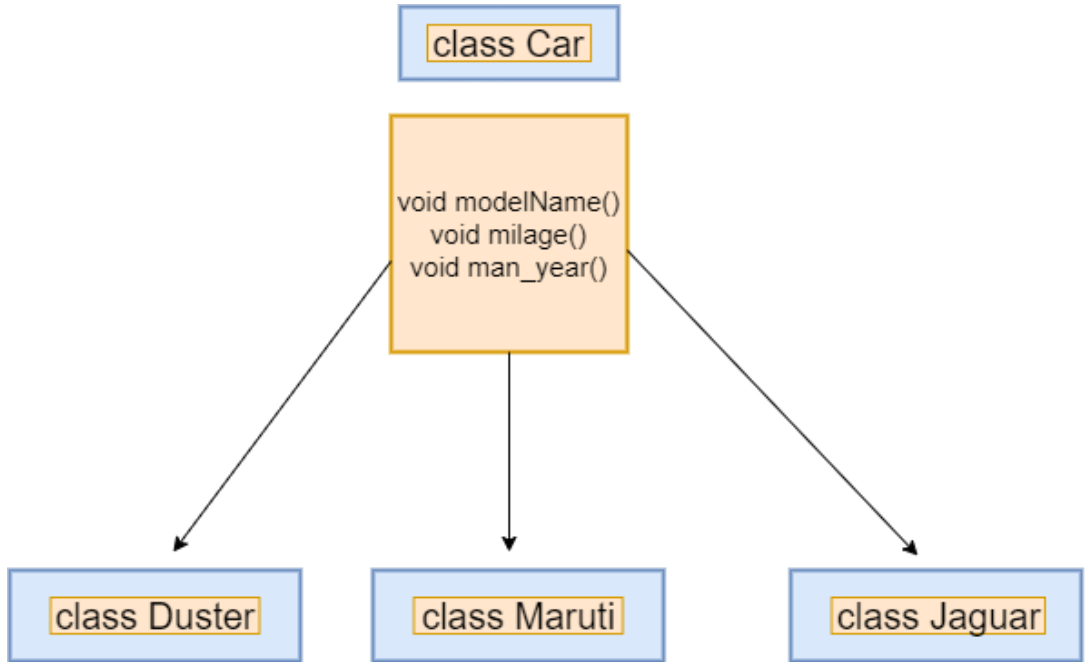
Dart inheritance is defined as the process of deriving the properties and characteristics of another class. It provides the ability to create a new class from an existing class. It is the most essential concept of the oops(Object-Oriented programming approach). We can reuse all the behavior and characteristics of the previous class in the new class.

• **Parent Class -** A class which is inherited by the other class is called **superclass** or **parent class**. It is also known as a **base class**.

• **Child Class -** A class which inherits properties from other class is called the child class. It is also known as the **derived class** or **subclass**.

```
class child_class extends parent_class
 {
  //body of child class
}
```

The child class inherits functions and variables, or properties of parent class using the extends keyword. It cannot inherit the parent class constructor



Types of Inheritance

1. Single Level Inheritance

In the single inheritance, a class is inherited by a single class or subclass is inherited by one parent class.

2. Multilevel Inheritance

In the multiple inheritance, a subclass is inherited by another subclass or creates the chaining of inheritance.

3. Hierarchical Inheritance

In the hierarchical inherence, two or more classes inherit a single class.

Note - Dart doesn't support multiple inheritance because it creates complexity in the program.

Dart super Keyword

The super keyword is used to denote the instant parent class object of the current child class. It is used to invoke superclass methods, superclass constructor in its child class. The super keyword's main objective is to remove the confusion between parent class and subclass with the same method name. It is also used to refer to the superclass properties and methods.

Usage of super Keyword

- When parent class and child class have members with the same name, then super keyword can be accessed data members of parent class in child class.
- It is used to access the parent class constructor in the child class.
- Using the super keyword, we can access the superclass method that is overridden by the subclass.

Using a super keyword with variables

This situation arises when the child class has the same name variables as superclass variables. So there is a chance of ambiguity for Dart compiler. Then super keyword can access the superclass variables in its child class. The syntax is given below.

```
Super.varName
```

```dart
// Super class Car
class Car {
  int speed = 180;
}

// sub class Bike extending Car
class Bike extends Car {
  int speed = 110;

  void display() {
    //print varible of the base class (Bike)
    print("The speed of car: $super.speed");
  }
}

void main() {
// Creating object of sub class
  Bike b = new Bike();
  b.display();
}
```

```
The speed of car: 180
```

Using the super keyword with parent class method

The super keyword is used to access the parent class method in child class. If the child class and parent class consist of the same name, then we can use the super keyword to use the parent class method in child class.

```
// Base class Super
class Super {
  void display() {
    print("This is the super class method");  } }
// Child class inherits Super
class Child extends Super {
  void display() {
    print("This is the child class");  }
   void message() {
    // will invoke or call current class display() method
    display();
    // will invoke or call parent class displa() method
    super.display(); } }
void main() {
   Child c = new Child();
  c.message();} // calling display() of Super
```

```
This is the child class
This is the super class method
```

Using super keyword with constructor

We can also use the super keyword to access the parent class constructor. The super keyword can call both parameterized and non-parameterized constructors depending on the situation.

```
// Base class called Parent
class Parent {
  Parent() {
    print("This is the super class constructor");
  }
}
// Child class Super
class Child extends Parent {
  Child() : super() // Calling super class constructor
  {
    print("This is the sub class constructor");
  }
}
void main() {
  // Creating object of sub class
  Child c = new Child();
}
```

```
This is the super class constructor
This is the sub class constructor
```

Dart polymorphism

The polymorphism is a combination of the two Greek words **poly,** which means **many** and morph means **different forms or shapes**. Together, polymorphism means the same entity can be used in various forms. In the programming aspect, the same method can be used in different classes. This technique makes programming more intuitive and more accessible.

For example - We have Shape class to define the shape of the object. The shape can be the circle, rectangle, square, straight line, etc. So here the goal is common, but the approach is different.

Method Overriding

When we declare the same method in the subclass, which is previously defined in the superclass is known as the method overriding. The subclass can define the same method by providing its own implementation, which is already exists in the superclass. The method in the superclass is called **method overridden,** and method in the subclass is **called method overriding**.

Method Overriding Example

We define two classes; first, is a subclass called **Student,** and the second is a superclass **College**. The Student subclass inherits the College superclass. The same method **void** `stu_details` in both classes is defined with the different implementation. The subclass has its own definition of the void `stu_details`.

```dart
class College {
  String? name;
  int? stdNo;
  void stu_details(name, stdNo) {
    this.name = name;
    this.stdNo = stdNo; }
  void display() {
    print("The student name:$name");
    print("The student stdno: $stdNo");
    print("The result is passed"); } }
class Student extends College {
  void stu_details(name, stdNo) {
    this.name = name;
    this.stdNo = stdNo;   }
  void show() {
    print("The student name:$name");
    print("The student stdno: $stdNo");
    print("The result is failed");  }  }
```

```dart
void main() {
  Student st = new Student();
st.stu_details("Mohamed", 101);
  st.show();
 College cg = new College();
  cg.stu_details("Ali", 102);
  cg.display();
}
```

```
The student name:Mohamed
The student stdno: 101
The result is failed
The student name:Ali
The student stdno: 102
The result is passed
```

Method Overriding using super Keyword

We can invoke the parent class method without creating its object. It can be done by using the super keyword in the subclass. The parent class data member can be accessed in the subclass by using the super keyword.

```
class Human {
  void run() {
    print("Human is running");
  }
}
class Man extends Human {
   void run() {
    // Accessing Parent class run() method in child class
    super.run();
    print("Boy is running");
  }
}
void main() {
  Man m = new Man();
  //This will call the child class version of eat()
  m.run(); }
```

```
Human is running
Boy is running
```

Advantage of method overriding

The main benefit of the method overriding is that the subclass can provide its own implementation to the same method as per requirement without making any changes in the superclass method. This technique is much when we want a subclass method to behave differently also with the same name.

Rules of Method overriding in Dart

1. The overriding method (the child class method) must be declared with the same configuration as the overridden method (the superclass method). The return type, list of arguments and its sequence must be the same as the parent class method.

2. The overriding method must be defined in the subclass, not in the same class.

3. The static and final method cannot be inherited in the subclass as they are accessible in their own class

4. The constructor of the superclass cannot be inherited in a subclass.

5. A method that cannot be inherited, then it cannot be overridden.

Dart Getters and Setters

Getters and setters are the special class method that is used to read and write access to an object's properties. The getter method is used to read the value of the variable or retrieve the value and setter method is used to set or initialize respective class fields. By default, all classes are associated with getter and setter method. However, we can override the default methods by defining getter and setter method explicitly.

Defining a getter

We can define the getters method by using the get keyword with no parameter a valid return type.

```
return_type get field_name{
}
```

Defining a setter

We can declare the setter method using the set keyword with one parameter and without return type.

```
set field_name( one parameter) {
}
```

```
class Employee {
  var empName = "Omer";
  var empAge = 24;
  var empSalary = 500;
  void set employeeName(String name) {
    this.empName = name;   }
  String get employeeName {
    return empName;   }
    void set employeeAge(int age) {
    if (age <= 18) {
      print("Employee Age should be greater than 18
Years.");
    } else {
      this.empAge = age;  }  }
  int get employeeAge {
    return empAge;  }
  void set employeeSalary(int salary) {
    if (salary <= 0) {
      print("Salary cannot be less than 0");
    } else {
      this.empSalary = salary;  }  }
  int get employeeSalary {
    return empSalary;  } }
```

```
void main() {
  Employee emp = new Employee();
  emp.employeeName = 'Ali';
  emp.employeeAge = 25;
  emp.employeeSalary = 2000;
  print("Employee's Name is :
${emp.employeeName}");
  print("Employee's Age is :
${emp.employeeAge}");
  print("Employee's Salary is :
${emp.employeeSalary}");
}
```

```
Employee's Name is : Ali
Employee's Age is : 25
Employee's Salary is : 2000
```

```
class Student {
  String stdName = "Khaled";
  String branch = "Math";
  int stdAge = 20;
  // getter method
void set std_name(String name) {
    this.stdName = name;   }
  String get std_name {
    return stdName;   }
    void set std_age(int age) {
    if (age <= 20) {
      print("Student age should be greater than 20");
    } else {
      this.stdAge = age;   }  }
  int get std_age {
    return stdAge;  }
  void set std_branch(String branch_name) {
    this.branch = branch_name;   }
  String get std_branch {
    return branch;  }}
```

```
void main() {
  Student std = new Student();
  std.std_name = 'John';
  std.std_age = 24;
  std.std_branch = 'Computer
Science';
  print("Student name is:
${std.std_name}");
  print("Student age is:
${std.std_age}");
  print("Student branch is:
${std.std_branch}");}
```

Student name is: John
Student age is: 24
Student branch is: Computer Science

When To Use Getter And Setter

• Use getter and setter when you want to restrict the access to the properties.

• Use getter and setter when you want to perform some action before reading or writing the properties.

• Use getter and setter when you want to validate the data before reading or writing the properties.