Object Oriented programming with Dart Programming Language

Daw A Mar Myo Thu

Lecturer

GUSTO University

Super in Dart

- Super is used to refer to the parent class.
- It is used to call the parent class's properties and methods.

Example : Super In Dart

In this example below, there is a class named Employee with a method named salary(). The salary() method is overridden in two child classes named Manager and Developer.

```
class Laptop {
  // Method
    void show() {
        print("Laptop show method");
    }
}
```

```
class MacBook extends Laptop {
    void show() {
        super.show(); // Calling the show method of the parent class
        print("MacBook show method");
    }
}
```

```
void main() {
  // Creating an object of the MacBook class
  MacBook macbook = MacBook();
  macbook.show();
}
```

POLYMORPHISM IN DART

- Poly means many and morph means forms.
- Polymorphism is the ability of an object to take on many forms.
- As humans, we have the ability to take on many forms. We can be a student, a teacher, a parent, a friend, and so on.
- Similarly, in object-oriented programming, polymorphism is the ability of an object to take on many forms.
- **Note**: In the real world, polymorphism is updating or modifying the feature, function, or implementation that already exists in the parent class.

Polymorphism By Method Overriding

- Method overriding is a technique in which you can create a method in the child class that has the same name as the method in the parent class.
- The method in the child class overrides the method in the parent class.

```
class ParentClass{
void functionName(){
  }
}
class ChildClass extends ParentClass{
@override
void functionName(){
  }
}
```

Example 1: Polymorphism By Method Overriding In Dart

In this example below, there is a class named Animal with a method named eat(). The eat() method is overridden in the child class named Dog.

```
class Animal {
  void eat() {
    print("Animal is eating");
  }
}
```

```
class Dog extends Animal {
  @override
  void eat() {
    print("Dog is eating");
  }
}
```

```
void main() {
  Animal animal = Animal();
  animal.eat();

  Dog dog = Dog();
  dog.eat();
}
```

Example 2: Polymorphism By Method Overriding In Dart

In this example below, there is a class named Vehicle with a method named run(). The run() method is overridden in the child class named Bus.

```
class Vehicle {
  void run() {
    print("Vehicle is running");
  }
}
```

```
class Bus extends Vehicle {
  @override
  void run() {
    print("Bus is running");
  }
}
```

```
void main() {
  Vehicle vehicle = Vehicle();
  vehicle.run();

  Bus bus = Bus();
  bus.run();
}
```

Example 3: Polymorphism By Method Overriding In Dart

In this example below, there is a class named Employee with a method named salary(). The salary() method is overridden in two child classes named Manager and Developer.

```
class Employee{
  void salary(){
    print("Employee salary is \$1000.");
  }
}
```

```
class Manager extends Employee{
  @override
  void salary(){
    print("Manager salary is \$2000.");
  }
}


class Developer extends Employee{
  @override
  void salary(){
    print("Developer salary is \$3000.");
  }
}
```

```
void main(){
  Manager manager=Manager();
  Developer developer=Developer();


  manager.salary();
  developer.salary();
}
```

Advantage Of Polymorphism In Dart

- Subclasses can override the behavior of the parent class.
- It allows us to write code that is more flexible and reusable.

ABSTRACT CLASS IN DART

- Abstract classes are classes that cannot be initialized.
- It is used to define the behavior of a class that can be inherited by other classes.
- An abstract class is declared using the keyword abstract.
- An abstract method is a method that is declared without an implementation.
- It is declared with a semicolon (;) instead of a method body.

```
abstract class ClassName {
  //Body of abstract class


  method1();
  method2();
}
```

Example 1: Abstract Class In Dart

In this example below, there is an abstract class Vehicle with two abstract methods start() and stop(). The subclasses Car and Bike implement the abstract methods and override them to print the message.

```
abstract class Vehicle {
  // Abstract method
  void start();
  // Abstract method
  void stop();
}

class Car extends Vehicle {
  // Implementation of start()
  @override
  void start() {
    print('Car started');
  }
```

```
  // Implementation of stop()
  @override
  void stop() {
    print('Car stopped');
  }
}

class Bike extends Vehicle {
  // Implementation of start()
  @override
  void start() {
    print('Bike started');
  }
```

```
  // Implementation of stop()
  @override
  void stop() {
    print('Bike stopped');
  }
}
void main() {
  Car car = Car();
  car.start();
  car.stop();
  Bike bike = Bike();
  bike.start();
  bike.stop();
}
```