Dart – Day6

Emp-id : 4781

- **Inheritance**

Inheritance allows a class to use the properties and methods of another class. The class that inherits is called the child (subclass), and the one being inherited from is the parent (superclass).

Example:

```
class Account
{
  void details()
  {
    print("This is a general bank account");
  }
}

class SavingsAccount extends Account
{
  void interestRate()
  {
    print("Savings Account Interest Rate: 6%");
  }
}

void main()
{
  var acc = SavingsAccount();
  acc.details();      // Inherited from Account
  acc.interestRate(); // Own method
}
```

- ## Inheritance – Super Constructor

A super constructor is used when the child class wants to call the constructor of its parent class. It helps initialize values from the parent.

Example:

```
class Employee
{
  String name;
  Employee(this.name); // parent constructor
}

class Manager extends Employee
{
  String department;
  Manager(String name, this.department) : super(name); // call parent constructor
}

void main()
{
  var m = Manager("Chandini", " Learning & Development ");
  print("${m.name} works in ${m.department} department");
  // Output: Chandini works in Learning & Development department
}
```

- ## Multilevel Inheritance

In multilevel inheritance, a class is derived from another derived class, forming a chain of inheritance.

Example:

```
class Device
{
  void powerOn()
  {
    print("Device is powering on");
  }
}
```

```
class Mobile extends Device
{
  void brand()
  {
    print("Brand: Samsung");
  }
}

class SmartPhone extends Mobile
{
  void features()
  {
    print("Smartphone Features: Touchscreen, Apps, Internet");
  }
}

void main()
{
  var phone = SmartPhone();
  phone.powerOn();  // from Device
  phone.brand();    // from Mobile
  phone.features(); // from SmartPhone
}
```

- ## Constructors

1. Parameterized Constructor with Named Parameters

Instead of passing values in order, you pass them with names using {}.

Example:

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

  Student({required this.name, required this.age}); // named parameters
```

```
  void show()
  {
    print("Name: $name, Age: $age");
  }
}

void main()
{
  var s = Student(name: "Chandini", age: 21);
  s.show();
}
```

2. Parameterized Constructor with Default Value

You can assign default values if no value is provided.

Example:

```
class Product
{
  String name;
  double price;

  Product({this.name = "Unknown", this.price = 0.0}); // default values

  void display()
  {
    print("Product: $name, Price: $price");
  }
}

void main()
{
  var p1 = Product(name: "Laptop", price: 50000);
  var p2 = Product(); // uses default values

  p1.display();
  p2.display();
```

```
}
```

3. Named Parameters Initialization

You can directly initialize values using this. inside the constructor.

Example:

```
class Employee
{
  String name;
  String role;

  Employee({required this.name, required this.role}); // initialization

  void info()
  {
    print("$name works as $role");
  }
}

void main()
{
  var e = Employee(name: "Sneha", role: "Developer");
  e.info();
}
```

4. Private Constructor

A constructor starting with _ makes it private (restricted to the same file).

Example:

```
class Logger
{
  Logger._internal(); // private constructor

  static final Logger _instance = Logger._internal();

  factory Logger()
```

```
  {
    return _instance; // always returns the same object
  }
}

void main()
{
  var log1 = Logger();
  var log2 = Logger();
  print(log1 == log2); // true (singleton pattern)
}
```

5. Getters and Setters

Used to access and update private variables safely.

Example:

```
class Account
{
  double _balance = 0; // private variable

  double get balance => _balance; // getter

  set balance(double amount)
  {   // setter
    if (amount >= 0) {
      _balance = amount;
    } else {
      print("Invalid amount");
    }
  }
}

void main()
{
  var acc = Account();
  acc.balance = 5000; // calls setter
  print(acc.balance); // calls getter
```

}