Dart – Day13

Emp-id : 4781

- **Abstract Class**

An abstract class is a class that cannot be instantiated directly.
It can contain both abstract methods (without body) and concrete methods (with body).
Subclasses must provide implementations for abstract methods.

Example :

```
abstract class Employee
{
  String name;
  Employee(this.name);
  void work();   // abstract method
  void details()
  {
    print("Employee: $name");
  }
}
class Developer extends Employee
{
  Developer(String name) : super(name);
  @override
  void work()
  {
    print("$name writes code.");
  }
```

```
}
class Designer extends Employee
{
  Designer(String name) : super(name);
  @override
  void work()
  {
    print("$name designs user interfaces.");
  }
}
void main()
{
  Employee e1 = Developer("Chandini");
  e1.details();
  e1.work();
  Employee e2 = Designer("Sneha");
  e2.details();
  e2.work();
}
```

- ## Factory Constructor

A factory constructor is a special constructor that does not always create a new object.
It can return an existing instance, a cached object, or even a different subtype.
This is useful for implementing singleton patterns or object caching.

Example :

```
class Student
{
```

```
  String name;
  int grade;
  Student._(this.name, this.grade);
  factory Student.pass(String name)
  {
    return Student._(name, 1);
  }

  factory Student.fail(String name)
  {
    return Student._(name, 0);
  }
  void showResult()
  {
    if (grade == 1)
    {
      print("$name has Passed");
    }
    else
    {
      print("$name has Failed");
    }
  }
}
void main()
{
  var s1 = Student.pass("Chandini");
```

```
  s1.showResult();
  var s2 = Student.fail("Sneha");
  s2.showResult();
}
```

- ## Interface

In Dart, there is no separate interface keyword.
Instead, any class can act as an interface if another class implements it.
When you implement a class as an interface:

→ You must override all its methods and properties (even if they already have bodies).
→ It is used as a contract that guarantees certain behaviors.

Example:

```
class Payment
{
  void pay(double amount)
  {
    print("Paying \$${amount}");
  }

  void refund(double amount)
  {
    print("Refunding \$${amount}");
  }
}
class UpiPayment implements Payment
{
  @override
```

```
  void pay(double amount)
  {
    print("Paid \$${amount} using UPI");
  }
  @override
  void refund(double amount)
  {
    print("Refunded \$${amount} via UPI");
  }
}
class CardPayment implements Payment
{
  @override
  void pay(double amount) {
    print("Paid \$${amount} using Credit/Debit Card");
  }
  @override
  void refund(double amount)
  {
    print("Refunded \$${amount} to Credit/Debit Card");
  }
}
void main()
{
  Payment upi = UpiPayment();
  upi.pay(500);
  upi.refund(200);
```

```
  Payment card = CardPayment();
  card.pay(1000);
  card.refund(300);
}
```