Abstraction: Abstraction is a core concept in object-oriented programming (OOP), and in Dart, it allows you to hide implementation details and only expose essential features of a class or interface. This helps you design cleaner, modular, and more maintainable code.

In Dart, abstraction is done using:

1. abstract classes
2. Interfaces (via abstract classes or implicit interfaces)
3. Mixins (partially abstract)

1) Using abstract class:

An abstract class cannot be instantiated. In dart don't declare the abstract method with abstract keyword. It can contain both:

- Abstract methods (without body)
- Concrete methods (with body)

Ex:
```
abstract class Animal
    {
       void sound();
       void breathe()
       {
          print("Breathing");
       }
   }

class Dog extends Animal
{
 @override
 void sound()
 {
   print("Bark");
 }
}

void main()
  {
      Dog d = Dog();
      d.sound();   // Bark
      d.breathe(); // Breathing
```

```
        }
```

2) **Interfaces in Dart:** In Dart, every class is an interface by default. So you can implement any class as an interface using implements.

Ex: class Vehicle

```
    {
        void move();
    }
 class Car implements Vehicle
{
  @override
  void move()
 {
    print("Car is moving");
   }
}


void main()
 {
  Car c = Car();
  c.move(); // Car is moving
}
```

Factory Constructors: A factory constructor in Dart is a special type of constructor that does not always create a new instance of a class. Instead, it can:

- Return an existing instance,
- Return a subtype,
- Perform logic before returning an object,
- Cache or reuse objects.

Syntax:

```
class ClassName
{
  factory ClassName()
  {
    // some logic
   return ClassName._internal();
  }

   ClassName._internal(); // private named constructor
}
```

Ex: class banking

```
    {

      String? accnum;

     String? accname;

     double? balance;

     banking._(this.accnum,this.accname,this.balance);

     factory banking.toCheck(String? accnum,String? accname,double? Balance )

    {

      if(balance!<100)

      {
```

```
        throw new Exception("Balance is insufficient");
      }
  else
 {
    return banking._( accnum, accname, balance);
  }

}

double deposit(double amt)
{
  return balance=balance!+amt;
}

double withDraw(double amt)
{
  if(balance!>amt)
  {
    balance=balance!-amt;
  }
  else
  {
    throw new Exception("Less balance !");
  }
```

```
  return balance!;
 }
}
void main()
{
   var a=banking.toCheck("a1", "john",,10000);
   print(a1.deposit(25000));
   print("Balance is ${a1.withDraw(10000)}");
}
```