* **enum**: kiểu liệt kê, là một lớp đặc biệt được sử dụng để đại diện cho một số lượng cố định các giá trị không đổi

Khai báo enum: enum Color {red,green, blue}

Mỗi giá trị trong enum đều có index getter, với giá trị đầu tiên có index = 0

* **async, await** : Thư viện Dart có hỗ trợ đầy đủ chức năng trả về các đối tượng Future hoặc Stream. Các chức năng này không đồng bộ
  + Sử dụng từ khóa async trước thân hàm để đánh dấu nó là bất đồng bộ
  + Để lấy dữ liệu từ Future có thể dùng await và await phải đặt trong async
  + Toàn bộ code trong hàm async sẽ chạy đồng bộ ngay lập tức đến khi gặp await và chờ kết quả của future sau await
* **late** - Sử dụng từ khóa late đối với các biến - các trường trong một lớp class hoặc các biến cao nhất - phải là kiểu non-nullable, nhưng chúng không thể gán giá trị ngay lập tức

class MonAn {

late String ten;

void setTen(String str) {

ten = str;

}

}

void main() {

final buaSang = MonAn();

buaSang.setTen('Bánh Mỳ');

print(buaSang.ten);

}

Khi sử dụng late trước một khai báo biến, điều đó sẽ giúp cho Dart biết rằng:

* + Biến này không cần kiểm tra phải có giá trị mặc định ngay
  + Bạn sẽ gán cho nó gía trị sau đó
  + Bạn sẽ đảm bảo rằng biến có một giá trị trước khi biến được sử dụng

Nếu khai báo một biến late và biến đó được sử dụng trước khi nó được gán giá trị, thì Dart sẽ báo lỗi

Việc sử dụng từ khóa late này còn được gọi là trì hoãn khởi tạo hoặc khởi tạo muộn, khởi tạo trễ (lazy initialization)

* **abstract** - trong dart chỉ có thể sử dụng từ khóa abstract trên các lớp. Các phương thức trừu tượng là các phương thức không có phần thân.
  + Khi muốn ghi đè một hoặc nhiều phương thức, chỉ cần sử dụng chú thích @override và đảm bảo sử dụng cùng một tên.
  + Trong Dart, có thể ghi đè các phương thức không trừu tượng nhưng đây được coi là một cách thực hành không tốt; Chỉ các phương thức trừu tượng nên được ghi đè
* **extends**
  + Đây là kế thừa OOP điển hình có thể được sử dụng khi bạn muốn thêm các tính năng mới trong một lớp con
  + Khi sử dụng class B extends A{}, bạn **KHÔNG** bị buộc phải ghi đè mọi phương thức của lớp A. Bạn có thể ghi đè bao nhiêu phương thức tùy ý
  + Dart chỉ hỗ trợ kế thừa đơn
* **implements**
  + Interfaces hữu ích khi bạn không muốn cung cấp triển khai các phương thức mà chỉ cung cấp API của chúng
  + Khi bạn sử dụng class B implements A{}, bạn phải ghi đè mọi phương thức của lớp A
  + Bạn có thể sử dụng implements với một hoặc nhiều lớp
* **with** - cho biết việc sử dụng mixin
* **mixin** - là cách sử dụng lại mã của một lớp trong nhiều phân cấp lớp, để sử dụng mixin, sử dụng từ khóa with theo sau là một hoặc nhiều tên mixin

mixin đề cập đến khả năng thêm các chức năng của một hoặc nhiều lớp vào lớp riêng của bạn mà không cần kế thừa từ các lớp đó. Các phương thức của các lớp đó bây giờ có thể được gọi trên lớp của bạn và mã bên trong các lớp đó sẽ được thực thi

* + Mixin hữu ích khi bạn cần chia sẻ code mà không sử dụng tính kế thừa
  + Khi sử dụng class B with A{}, bạn importing mọi phương thức của mixin A vào lớp B. Theo tùy chọn, việc sử dụng mixin có thể bị giới hạn ở một loại nhất định bằng cách sử dụng từ khóa on
  + Bạn có thể sử dụng with một hoặc nhiều mixin
* **as** - Sử dụng toán tử as để cast một đối tượng đến một kiểu cụ thể khi và chỉ khi bạn biết chắc chắn rằng đối tượng thuộc kiểu đó
* **class** - theo sau là một tên lớp được sử dụng để định nghĩa một lớp. Tất cả các trường và hàm được bao bởi cặp dấu ngoặc nhọn ({}). syntax

class ClassName{

<fields>

<getters/setters>

<constructor>

<function>

}

* **extension** - extension method là một cách để thêm chức năng vào các thư viện hiện có. Có thể sử dụng các Extension methods mà không cần biết nó.
* **factory** - Dart cung cấp cho chúng ta factory keyword để tạo ra 1 constructor đặc biệt, khi sử dụng constructor này để tạo đối tượng nó sẽ không tạo ra một đối tượng mới nếu nó thấy đã có một đối tượng có sẵn rồi
* **yield** - thêm một giá trị vào output stream của hàm async\*. Nó tương tự như return nhưng không không chấm dứt hàm