

# LẬP TRÌNH TRÊN THIẾT BỊ DI ĐỘNG

## Bài 2. Giới thiệu về ngôn ngữ lập trình DART

ThS. Lê Nhật Tùng

- ❶ 2.1. Giới thiệu ngôn ngữ lập trình DART
- ❷ 2.2. Cài đặt môi trường phát triển
- ❸ 2.3. Lập trình DART

1 2.1. Giới thiệu ngôn ngữ lập trình DART

2 2.2. Cài đặt môi trường phát triển

3 2.3. Lập trình DART

# Dart là gì?

- Dart là một ngôn ngữ lập trình mạnh mẽ, mã nguồn mở được phát triển bởi Google.
- Được thiết kế để xây dựng các ứng dụng hiện đại với hiệu suất cao, Dart được sử dụng chủ yếu trong phát triển giao diện người dùng cho các ứng dụng di động, web và máy tính để bàn.
- Ngôn ngữ Dart đặc biệt nổi bật khi kết hợp với Flutter để tạo ra các ứng dụng đa nền tảng với giao diện người dùng đẹp mắt và mượt mà.
- Thường được trang bị các hệ điều hành mạnh mẽ, màn hình cảm ứng và nhiều cảm biến khác nhau để cung cấp trải nghiệm người dùng.

# Lịch sử ngôn ngữ lập trình DART

- **2011:** Được Google giới thiệu lần đầu tại hội nghị GOTO
- **2013:** Phát hành Dart 1.0 - Mục tiêu thay thế JavaScript
- **2015:** Dart 1.9 giới thiệu hỗ trợ async/await
- **2018:** Ra mắt Dart 2.0
  - Hệ thống kiểu mạnh mẽ; Bảo mật null an toàn; Tối ưu hóa cho phát triển phía client
- **2021:** Phát hành Dart 2.14 với các tính năng mới
  - Nâng cao hệ thống kiểu; Cải thiện công cụ phát triển; Tích hợp tốt hơn với Flutter
- **Hiện tại:** Là ngôn ngữ chính cho framework Flutter



# Các phiên bản quan trọng của DART

- **Dart 2.0 (2018)**

- Cải tiến cú pháp và hiệu suất
- Nâng cao khả năng biên dịch
- Dầu mốc quan trọng để trở thành ngôn ngữ đa năng

- **Dart 2.12 (2021)**

- Giới thiệu null safety
- Tăng cường bảo mật mã nguồn
- Cải thiện hiệu quả thực thi

- **Dart 3.0 (2023)**

- Pattern matching
- Records
- Class modifiers

- **Dart 3.4 (2024)**

- Tính năng Macros
- Hỗ trợ WebAssembly (Wasm)
- Nâng cao hiệu suất tổng thể

# Ưu điểm của ngôn ngữ lập trình DART

- **Hiệu năng cao**

- Biên dịch trực tiếp thành mã máy
- Tối ưu hóa thời gian chạy

- **Tính năng hiện đại**

- Hỗ trợ lập trình bất đồng bộ
- Null safety
- Pattern matching

- **Hệ sinh thái phong phú**

- Framework Flutter
- Pub.dev package manager
- DevTools mạnh mẽ

- **Dễ học và sử dụng**

- Cú pháp quen thuộc với lập trình viên
- Tài liệu phong phú
- Cộng đồng hỗ trợ tích cực

1 2.1. Giới thiệu ngôn ngữ lập trình DART

2 2.2. Cài đặt môi trường phát triển

3 2.3. Lập trình DART



- **Thành phần chính**

- Trình biên dịch Dart (dart)
- Máy ảo Dart VM
- Thư viện chuẩn (dart:core)
- Công cụ phát triển (dartanalyzer)

- **Công cụ đi kèm**

- pub: Quản lý package
- dart2js: Biên dịch sang JavaScript
- dartfmt: Format code
- dartdoc: Tạo tài liệu

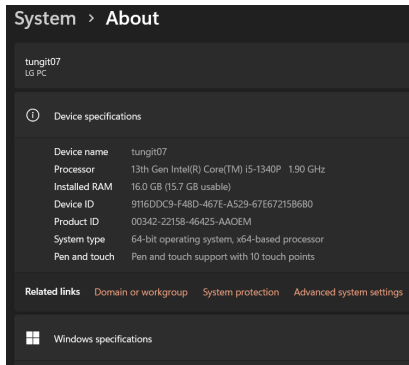
- **DevTools**

- Debugger
- Performance profiler
- Memory analyzer
- Widget inspector (cho Flutter)

# Cấu hình PATH cho Dart SDK - Bước 1

## • Bước 1:

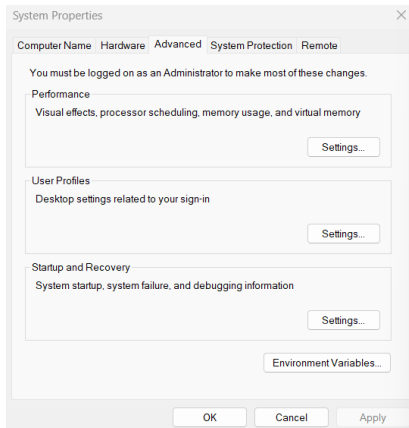
- Nhấn Windows + R
- Gõ "sysdm.cpl"
- Chọn tab "Advanced"



# Cấu hình PATH cho Dart SDK - Bước 2

## • Bước 2:

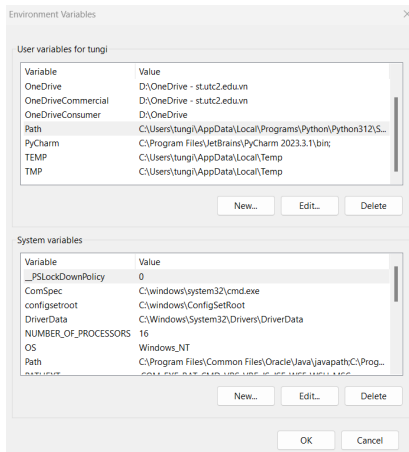
- Nhấn "Environment Variables"
- Tìm mục "System variables"



# Cấu hình PATH cho Dart SDK - Bước 3

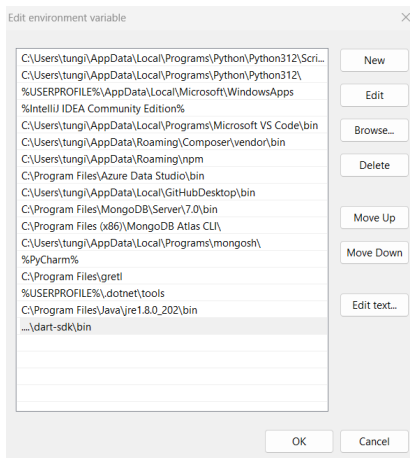
## • Bước 3:

- Tìm biến PATH
- Nhấn "Edit"



# Cấu hình PATH cho Dart SDK - Bước 4

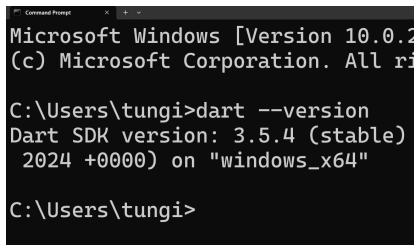
- Kiểm tra cấu hình:
  - Thêm đường dẫn:  
[...]/dart-sdk/bin



# Cấu hình PATH cho Dart SDK - Hoàn thành

## • Xác nhận cài đặt:

- Mở Command Prompt mới
- Gõ lệnh: `dart -version`
- Dart SDK đã sẵn sàng sử dụng
- Có thể bắt đầu lập trình



```
Microsoft Windows [Version 10.0.22H2]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tungi>dart --version
Dart SDK version: 3.5.4 (stable)
2024-01-10T14:00:00.000 on "windows_x64"

C:\Users\tungi>
```

# DartPad - Công cụ lập trình Dart trực tuyến

- **Giới thiệu**

- Công cụ viết code Dart trực tuyến miễn phí
- Truy cập tại: <https://dartpad.dev>
- Không cần cài đặt, chạy trực tiếp trên trình duyệt

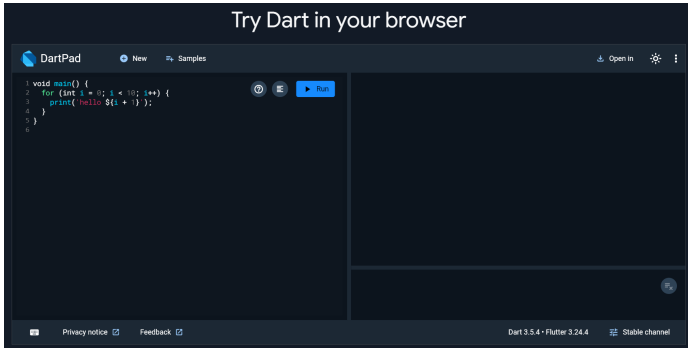
- **Tính năng chính**

- Soạn thảo code với syntax highlighting
- Tự động hoàn thành code
- Phát hiện lỗi cú pháp thời gian thực
- Chạy và xem kết quả ngay lập tức

- **Hỗ trợ**

- Code Dart thuần
- Flutter (UI development)
- Chia sẻ code qua URL

# Giao diện DartPad





- 1 2.1. Giới thiệu ngôn ngữ lập trình DART
- 2 2.2. Cài đặt môi trường phát triển
- 3 2.3. Lập trình DART

# Khái niệm cơ bản trong Dart

- **Biến (Variables)**

- Hỗ trợ kiểu dữ liệu nguyên thủy (int, double, bool)
- Kiểu phức tạp (List, Map)
- Khai báo với từ khóa var, final hoặc const

- **Hàm (Functions)**

- Khai báo bằng từ khóa function
- Hỗ trợ hàm không tên (anonymous functions)
- Hàm mũi tên (arrow functions)

- **Lớp và Đối tượng (Classes and Objects)**

- Ngôn ngữ lập trình hướng đối tượng
- Mọi thứ trong Dart đều là đối tượng
- Class định nghĩa thuộc tính và phương thức

- **Thừa kế (Inheritance)**

- Cho phép tạo lớp con kế thừa từ lớp cha
- Tái sử dụng và tổ chức mã nguồn tốt hơn

- **Mixin**

- Kế thừa từ nhiều lớp khác nhau
- Chia sẻ hành vi giữa các lớp

# Quy tắc đặt tên trong Dart

- Quy tắc đặt tên

- camelCase cho tên biến và hàm
- PascalCase cho tên lớp

```
1 // Ten bien (camelCase)
2 String firstName;
3 int maxCount;
4 double totalPrice;
5
6 // Ten ham (camelCase)
7 void calculateTotal() { }
8 int getUserAge() { }
9
10 // Ten lop (PascalCase)
11 class UserProfile { }
12 class DatabaseHelper { }
13 class PaymentService { }
```

# Dấu chấm phẩy trong Dart

- **Dấu chấm phẩy**

- Kết thúc mỗi câu lệnh bằng dấu chấm phẩy
- Bắt buộc để tránh lỗi cú pháp

```
1 void main() {  
2   String name = "John";           // Dấu chấm phẩy  
3   int age = 25;                   // Dấu chấm phẩy  
4  
5   if (age >= 18) {  
6     print("Hello $name");         // Dấu chấm phẩy  
7   }                               // Không cần dấu  
                                   // chấm phẩy  
8 }                                 // Không cần  
                                   // dấu chấm phẩy
```

# Dấu nháy trong Dart

- **Dấu nháy**

- Sử dụng {} cho khối mã
- Trong hàm, điều kiện if, vòng lặp

```
1 // Hàm với dấu nháy
2 void kiểmTraTuoi() {
3     int tuoi = 20;
4
5     // Điều kiện if với dấu nháy
6     if (tuoi >= 18) {
7         print("Đã trưởng thành");
8     }
9
10    // Vòng lặp với dấu nháy
11    for (int i = 0; i < 5; i++) {
12        print(i);
13    }
14 }
```

# Thụt lề trong Dart

- **Thụt lề**

- Sử dụng khoảng trắng hoặc tab
- Giúp code dễ đọc và bảo trì

```
1 class NguoIDung {  
2     String ten;  
3     int tuoi;  
4  
5     void hienThiThongTin() {  
6         if (tuoi >= 18) {  
7             print("Ten: $ten");  
8             print("Tuoi: $tuoi");  
9         } else {  
10            print("Chua du tuoi");  
11        }  
12    }  
13 }
```

# Chú thích trong Dart

## • Dấu chú thích

- Chú thích đơn dòng (//)
- Chú thích đa dòng (/\* \*/)
- Documentation comments (///)

```
1 // Chu thích mot dong
2 int tuoi = 25;
3
4 /* Chu thích nhieu dong
5    Mo ta chi tiet hon ve
6    chuc nang cua doan code */
7
8 /// Documentation comment
9 /// Mo ta cho class hoac phuong thuc
10 class SinhVien {
11     String ten;
12     // TODO: Them cac thuoc tinh khac
13 }
```

# Khai báo biến trong Dart

- Biến trong Dart được sử dụng để lưu trữ các tham chiếu tới các đối tượng.
- **Cách khai báo biến**
  - Sử dụng var để Dart tự suy luận kiểu
  - Khai báo với kiểu cụ thể
  - Khai báo với Object

```
1 // Tu suy luận kiểu
2 var ten = 'Tung';
3
4 // Khai báo kiểu cụ thể
5 String hoTen = 'Tung';
6
7 // Khai báo với Object
8 Object tenNguoiDung = 'Tung';
```



# Null Safety trong Dart

## • Tính năng Null Safety

- Ngăn chặn lỗi Null Dereference
- Phát hiện lỗi tại thời điểm biên dịch
- Sử dụng ? để cho phép null
- Lưu ý: Hàm assert() được sử dụng để kiểm tra các điều kiện trong quá trình phát triển (development). Nếu điều kiện trong assert() là false, chương trình sẽ ném ra lỗi và dừng lại.

```
1 // Kiểu dữ liệu có thể null
2 String? ten;
3 int? soDienThoai;
4
5 // Kiểu dữ liệu không thể null
6 String hoTen = 'Tung';
7 int tuoi = 25;
8
9 // Kiểm tra null
10 int? soLuong;
11 assert(soLuong == null);
```

# Biến late trong Dart

- **Từ khóa late**

- Khởi tạo biến trước khi sử dụng
- Hỗ trợ Lazy Initialization

```
1 // Khai báo late
2 late String moTa;
3
4 void main() {
5     moTa = 'Truong HUTECH!';
6     print(moTa);
7 }
8
9 // Lazy Initialization
10 late String nhietDo = docNhietKe();
```

# Biến final và const trong Dart

## • Final và Const

- Biến final chỉ được gán một lần và không thể thay đổi giá trị sau khi gán.
- Biến const là hằng số tại thời gian biên dịch và không thể thay đổi giá trị.

```
1 // Biến final
2 final name = 'Tung'; // Biến final
3 name = 'Cuong'; // Lỗi: không thể thay đổi giá
   trị của biến final
4 // Biến const
5 const int namHienTai = 2024;
6 const double PI = 3.14159;
7 // Sử dụng const phức tạp
8 const Object i = 3;
9 const list = [i as int];
10 const map = {if (i is int) i: 'int'};
11 const set = {if (list is List<int>) ...list};
```

# Ví dụ: Khai báo và sử dụng biến - Phần 1

```
1 import 'dart:io';
2
3 void main() {
4     // Khai báo biến cơ bản
5     var ten = 'Tung';      // Tu suy luận kiểu
6                             String
7     int tuoi = 25;          // Khai báo rõ ràng
8                             kiểu
9
10    // Sử dụng null safety
11    String? tenNullable;
12    tenNullable = null;     // Cho phép null
```

## Ví dụ: Khai báo và sử dụng biến - Phần 2

```
1 // Sử dụng biến late
2 late String moTa;
3 moTa = 'Lập trình Dart';
4
5 // Sử dụng final và const
6 final String quocGia = 'Việt Nam';
7 const int nam = 2024;
```

## Ví dụ: Khai báo và sử dụng biến - Phần 3

```
1      // In gia tri bien
2      print('Ten: $ten');
3      print('Tuoi: $tuoi');
4      print('Mo ta: $moTa');
5      print('Quoc gia: $quocGia');
6      print('Nam: $nam');
7
8      // Kiem tra bien nullable
9      int? soLuong;
10     assert(soLuong == null);
11 }
```

# Toán tử tăng và giảm trong Dart

- **Tiền tố (Prefix)**

- $++a$ : Tăng  $a$  trước khi sử dụng
- $-a$ : Giảm  $a$  trước khi sử dụng

- **Hậu tố (Postfix)**

- $a++$ : Sử dụng  $a$  trước khi tăng
- $a--$ : Sử dụng  $a$  trước khi giảm

# Toán tử tiền tố tăng (++a)

```
1 void main() {  
2     int a;  
3     int b;  
4  
5     // Tang truoc (prefix increment)  
6     a = 0;  
7     b = ++a;    // Tang a truoc khi gan cho b  
8     assert(a == b);    // 1 == 1  
9  
10    print('a = $a');    // In ra: a = 1  
11    print('b = $b');    // In ra: b = 1  
12 }
```



# Toán tử hậu tố tăng (a++)

```
1 void main() {  
2     int a;  
3     int b;  
4  
5     // Tang sau (postfix increment)  
6     a = 0;  
7     b = a++;    // Gan a cho b roi moi tang a  
8     assert(a != b);    // 1 != 0  
9  
10    print('a = $a');    // In ra: a = 1  
11    print('b = $b');    // In ra: b = 0  
12 }
```

# Toán tử tiền tố giảm ( $-a$ )

```
1 void main() {  
2     int a;  
3     int b;  
4  
5     // Giảm trước (prefix decrement)  
6     a = 0;  
7     b = --a;    // Giảm a trước khi gán cho b  
8     assert(a == b);    // -1 == -1  
9  
10    print('a = $a');    // In ra: a = -1  
11    print('b = $b');    // In ra: b = -1  
12 }
```

# Toán tử hậu tố giảm (a--)

```
1 void main() {
2     int a;
3     int b;
4
5     // Giảm sau (postfix decrement)
6     a = 0;
7     b = a--;    // Gán a cho b rồi mới giảm a
8     assert(a != b);    // -1 != 0
9
10    print('a = $a');    // In ra: a = -1
11    print('b = $b');    // In ra: b = 0
12 }
```

# Câu hỏi trắc nghiệm - Toán tử tăng giảm

## Câu hỏi

Xem đoạn code sau và chọn giá trị cuối cùng của a:

```
1 void main() {  
2     int a = 5;  
3     print(a++);      // Step 1  
4     print(++a);      // Step 2  
5     print(a--);      // Step 3  
6     print(--a);      // Step 4  
7     print(a);        // Step 5  
8 }
```

- A) 5, 7, 7, 5, 5 | B) 5, 7, 7, 5, 4  
C) 6, 7, 6, 5, 5 | D) 5, 6, 6, 4, 4

# Câu hỏi trắc nghiệm - Toán tử tăng giảm

## Câu hỏi

Xem đoạn code sau và chọn giá trị cuối cùng của a:

```
1 void main() {  
2     int a = 5;  
3     print(a++);      // Step 1  
4     print(++a);      // Step 2  
5     print(a--);      // Step 3  
6     print(--a);      // Step 4  
7     print(a);        // Step 5  
8 }
```

A) 5, 7, 7, 5, 5 | B) 5, 7, 7, 5, 4

C) 6, 7, 6, 5, 5 | D) 5, 6, 6, 4, 4

## Đáp án

**A) 5, 7, 7, 5, 5**

# Toán tử so sánh và quan hệ trong Dart

Toán tử	Ý nghĩa
$==$	Bằng
$!=$	Không bằng
$>$	Lớn hơn
$<$	Nhỏ hơn
$>=$	Lớn hơn hoặc bằng
$<=$	Nhỏ hơn hoặc bằng

# Ví dụ về toán tử so sánh

```
1 void main() {  
2     // So sánh bằng  
3     assert(2 == 2);    // true  
4  
5     // So sánh khác  
6     assert(2 != 3);    // true  
7  
8     // So sánh lớn hơn  
9     assert(3 > 2);     // true  
10  
11    // So sánh nhỏ hơn  
12    assert(2 < 3);     // true  
13  
14    // So sánh lớn hơn hoặc bằng  
15    assert(3 >= 3);    // true  
16  
17    // So sánh nhỏ hơn hoặc bằng  
18    assert(2 <= 3);    // true  
19 }
```

# Toán tử kiểm tra kiểu trong Dart

Toán tử	Ý nghĩa
as	Ép kiểu
is	Kiểm tra kiểu
!is	Kiểm tra không phải kiểu



# Ví dụ về toán tử kiểm tra kiểu

```
1 void main() {  
2     Object obj = 'Hello';  
3  
4     // Kiểm tra kiểu String  
5     if (obj is String) {  
6         print('obj là String');  
7     }  
8  
9     // Kiểm tra không phải kiểu int  
10    if (obj !is int) {  
11        print('obj không phải là int');  
12    }  
13  
14    // Ép kiểu  
15    String str = obj as String;  
16    print(str.toUpperCase());  
17 }
```

# Lưu ý khi sử dụng toán tử kiểu

- **Toán tử as**

- Có thể ném ra Exception nếu ép kiểu thất bại
- Nên kiểm tra với is trước khi ép kiểu

- **Toán tử is**

- Kiểm tra an toàn hơn as
- Tự động ép kiểu trong scope if

- **Toán tử !is**

- Ngắn gọn hơn !(obj is Type)
- Thường dùng trong điều kiện phủ định

# Toán tử gán trong Dart

Toán tử	Ý nghĩa
=	Gán giá trị
?=	Gán nếu null
+=, -=, *=, /=	Gán kết hợp

# Toán tử gán cơ bản

```
1 void main() {  
2     // Gán giá trị cơ bản  
3     var a = 2;  
4     print('a = $a');    // 2  
5  
6     // Gán neu null  
7     int? b;  
8     b ??= 5;    // Gán 5 vì b đang null  
9     print('b = $b');    // 5  
10  
11     b ??= 10;    // Không gán vì b không null  
12     print('b = $b');    // Vẫn là 5  
13 }
```

# Toán tử gán kết hợp

```
1 void main() {  
2     var n = 5;  
3  
4     // Gán và cộng  
5     n += 3;      // n = n + 3  
6     print('n = $n');    // 8  
7  
8     // Gán và trừ  
9     n -= 2;      // n = n - 2  
10    print('n = $n');    // 6  
11  
12    // Gán và nhân  
13    n *= 2;       // n = n * 2  
14    print('n = $n');    // 12  
15  
16    // Gán và chia  
17    n ~/= 3;      // n = n ~/ 3 (chia lấy phần  
18                    nguyên)  
19    print('n = $n');    // 4
```

# Ví dụ thực tế với toán tử gán

```
1 void main() {
2     int diemToan = 8;
3     int diemVan = 7;
4     int tongDiem = 0;
5     // Cong diem tung mon
6     tongDiem += diemToan;
7     tongDiem += diemVan;
8     // Tinh diem trung binh
9     double diemTB = tongDiem / 2;
10    // Gan diem dat/khong dat
11    String? ketQua;
12    ketQua ??= 'Chua xet';
13
14    if (diemTB >= 5) {
15        ketQua = 'Dat';
16    }
17    print('Diem TB: $diemTB');
18    print('Ket qua: $ketQua');
19 }
```

# Câu hỏi về toán tử gán

A)  $x = 10, y = 30$

B)  $x = 10, y = 50$

C)  $x = 10, y = 15$

D)  $x = 10, y = 10$

# Câu hỏi về toán tử gán

A)  $x = 10, y = 30$

B)  $x = 10, y = 50$

C)  $x = 10, y = 15$

D)  $x = 10, y = 10$

**Đáp án:** A)  $x = 10, y = 30$  - Step 1:  $x = 15$  - Step 2:  $x = 30$  - Step 3:  $y = 30$  (vì  $y$  là null) - Step 4:  $x = 10$  - Step 5:  $y$  không thay đổi (vì đã có giá trị)



# Toán tử thao tác bit trong Dart

- & AND bit
- | OR bit
- ^ XOR bit
- ~ Phủ định bit
- << Dịch trái
- >> Dịch phải

# AND và OR bit

```
1 void main() {  
2     final value = 0x22;    // 34 trong he 10  
3     final bitmask = 0x0f; // 15 trong he 10  
4  
5     // AND bit (&)  
6     print((value & bitmask).toRadixString(16));  
7     // In: 02  
8  
9     // OR bit (|)  
10    print((value | bitmask).toRadixString(16));  
11    // In: 2f  
12 }
```

# XOR và NOT bit

```
1 void main() {  
2     final value = 0x22;  
3     final bitmask = 0x0f;  
4  
5     // XOR bit (^)  
6     print((value ^ bitmask).toRadixString(16));  
7     // In: 2d  
8  
9     // NOT và AND bit (~)  
10    print((value & ~bitmask).toRadixString(16));  
11    // In: 20  
12 }
```

# Toán tử điều kiện trong Dart

## Các loại toán tử điều kiện

- `condition ? expr1 : expr2`
  - Nếu `condition` đúng, trả về `expr1`
  - Nếu `condition` sai, trả về `expr2`
- `expr1 ?? expr2`
  - Nếu `expr1` khác `null`, trả về `expr1`
  - Nếu `expr1` là `null`, trả về `expr2`

```
1 void main() {  
2     // Toan tu dieu kien (?)  
3     bool isPublic = true;  
4     var visibility = isPublic ? 'public' :  
5         'private';  
6  
7     // Toan tu null (??)  
8     String? name;  
9     var displayName = name ?? 'Khach';  
}
```

# Khi nào nên dùng toán tử điều kiện

- **Sử dụng ?: khi:**

- Cần lựa chọn nhanh giữa hai giá trị
- Thay thế cho if-else ngắn
- Gán giá trị dựa trên điều kiện

- **Sử dụng ?? khi:**

- Cần giá trị mặc định cho biến null
- Xử lý dữ liệu từ API có thể null
- Khởi tạo giá trị an toàn

```
1 void main() {  
2     String layTenHienThi(String? ten, bool  
3         daLogin) {  
4         return daLogin ? (ten ?? 'Nguoi dung') :  
5             'Khach';  
6     }  
7 }
```

# Toán tử Cascade (..) trong Dart

- **Toán tử Cascade (..)** cho phép:

- Thực hiện nhiều thao tác trên cùng một đối tượng
- Tránh việc phải lặp lại tên đối tượng
- Viết code ngắn gọn và rõ ràng hơn

```
1 void main() {  
2     // Không dùng cascade  
3     var list = [1, 2];  
4     list.add(3);  
5     list.add(4);  
6     list.remove(2);  
7  
8     // Dùng cascade  
9     var list2 = [1, 2]  
10    ..add(3)  
11    ..add(4)  
12    ..remove(2);  
13 }
```

# Ví dụ thực tế với List và StringBuffer

```
1 void main() {  
2     // Ví dụ với List  
3     final numbers = <int>[]  
4         ..add(1)  
5         ..addAll([2, 3, 4])  
6         ..remove(2)  
7         ..sort();  
8  
9     // Ví dụ với StringBuffer  
10    final buffer = StringBuffer()  
11        ..write('Xin '  
12        ..write('chao '  
13        ..writeln('ban');  
14  
15    print(numbers);    // [1, 3, 4]  
16    print(buffer);    // Xin chao ban  
17 }
```

# Siêu dữ liệu trong Dart

- **Metadata (@)** cho phép:

- Cung cấp thông tin bổ sung về code
- Được đặt trước declarations
- Bắt đầu bằng ký tự @

```
1 // Metadata có sẵn trong Dart
2 @deprecated
3 void oldMethod() {
4     print('Phương thức cũ, không nên dùng');
5 }
6
7 @override
8 void toString() {
9     return 'Đây là lớp con';
10 }
```



# Các metadata phổ biến

```
1 class Animal {
2     void makeSound() {
3         print('Some sound');
4     }
5 }
6
7 class Cat extends Animal {
8     @override // Ghi đè phương thức
9     void makeSound() {
10         print('Meow');
11     }
12
13     @deprecated // Đánh dấu phương thức cũ
14     void oldMethod() {
15         print('Không nên dùng nữa');
16     }
17
18     @pragma('vm:prefer-inline') // Gợi ý tối ưu
19     void doSomething() {
```



# Tự định nghĩa Metadata

```
1 // Định nghĩa metadata
2 class Todo {
3     final String who;
4     final String what;
5
6     const Todo(this.who, this.what);
7 }
8
9 // Sử dụng metadata tự định nghĩa
10 @Todo('Nam', 'Can kiểm tra lại phương thức này')
11 void doSomething() {
12     print('Doing something...');
13 }
```

# Thư viện trong Dart

- **Tổng quan:**

- Giúp mã nguồn mô-đun hóa và dễ chia sẻ
- Bảo vệ mã nguồn (định danh bắt đầu bằng dấu gạch dưới)
- Mỗi tệp Dart là một thư viện

- **Cách sử dụng:**

- Sử dụng từ khóa import
- Truy cập lớp, hàm từ thư viện

```
1 // Thu vien tích hợp
2 import 'dart:html';
3
4 // Thu vien ngoại
5 import 'package:test/test.dart';
```

# Đặt tên tiền tố cho thư viện

- **Mục đích:**

- Tránh xung đột tên giữa các thư viện
- Phân biệt nguồn gốc của các thành phần

```
1 // Khai bao thu vien
2 import 'package:lib1/lib1.dart';
3 import 'package:lib2/lib2.dart' as lib2;
4
5 void main() {
6     // Su dung Element tu lib1
7     Element element1 = Element();
8
9     // Su dung Element tu lib2
10    lib2.Element element2 = lib2.Element();
11 }
```

# Import một phần thư viện

- **Từ khóa:**

- show: Chỉ import các thành phần cụ thể
- hide: Loại trừ các thành phần không cần thiết

```
1 // Chỉ import foo
2 import 'package:lib1/lib1.dart' show foo;
3
4 // Import tất cả trừ foo
5 import 'package:lib2/lib2.dart' hide foo;
6
7 // Import nhiều thành phần
8 import 'package:lib3/lib3.dart'
9     show foo, bar, baz;
```

# Lazy Loading

- **Đặc điểm:**

- Tải thư viện khi cần thiết
- Giảm thời gian khởi động
- Tối ưu hóa tài nguyên

```
1 // Khai bao lazy loading
2 import 'package:greetings/hello.dart'
3     deferred as hello;
4
5 Future<void> greet() async {
6     await hello.loadLibrary(); // Load khi can
7     hello.printGreeting();
8 }
9
10 void main() async {
11     await greet(); // Thu vien se duoc tai
12 }
```

- **Các điểm cần nhớ:**

- URI là tham số bắt buộc cho import
- Thư viện tích hợp dùng tiền tố dart:
- Thư viện ngoài dùng tiền tố package:
- `loadLibrary()` chỉ tải thư viện một lần

- **Thực hành tốt:**

- Sử dụng tiền tố khi có khả năng xung đột
- Import có chọn lọc để tối ưu code
- Sử dụng lazy loading cho thư viện lớn

- **Định nghĩa:**

- Các từ được ngôn ngữ Dart dành riêng
- Không thể sử dụng làm định danh (trừ trường hợp đặc biệt)
- Giúp xác định cấu trúc và chức năng của code



- **Loại 1: Hạn chế theo ngữ cảnh**
  - await
  - yield
- **Loại 2: Không dùng cho tên kiểu dữ liệu**
  - as, dynamic, Function, implements
  - import, interface, mixin, part
  - required, set, static, typedef
- **Loại 3: Không hạn chế**
  - async, base, hide, of, on
  - sealed, show, sync, when

# Từ khóa thường dùng

```
1 // Khai bao lop
2 abstract class Animal {
3     void makeSound();
4 }
5
6 // Su dung async/await
7 Future<void> getData() async {
8     await Future.delayed(Duration(seconds: 1));
9     return;
10 }
11
12 // Su dung final va const
13 final int count = 10;
14 const double PI = 3.14;
15
16 // Su dung control flow
17 if (condition) {
18     // code
19 } else {
```

- **Quy tắc chung:**

- Tránh sử dụng từ khóa làm định danh
- Chú ý ngữ cảnh sử dụng
- Tuân thủ quy tắc hạn chế của từng loại

- **Thực hành tốt:**

- Đặt tên rõ ràng, tránh nhầm lẫn
- Tham khảo tài liệu chính thức
- Sử dụng IDE hỗ trợ syntax highlighting

# Kiểu dữ liệu trong Dart

- **Các kiểu dữ liệu cơ bản:**

- Numbers (int, double)
- String
- Boolean
- Lists
- Sets
- Maps
- Runes

- **Đặc điểm:**

- Hệ thống kiểu phong phú và đa dạng
- Mạnh mẽ và hiệu quả
- Hỗ trợ kiểu dữ liệu tĩnh và động

# Kiểu số trong Dart

- **int**: Số nguyên không có phần thập phân
- **double**: Số thực có phần thập phân
- **num**: Có thể chứa cả int và double

```
1 void main() {  
2   // So nguyen (int)  
3   var x = 10;  
4   var hex = 0xDEADBEEF;  
5   // So thuc (double)  
6   var y = 1.1;  
7   var exponents = 1.42e5;  
8   // Kieu num  
9   num z = 1;  
10  z += 2.5; // z chua ca int va double  
11  // Chuyen doi  
12  var one = int.parse('1');  
13  var onePointOne = double.parse('1.1');  
14  String piAsString =  
    3.14159.toStringAsFixed(2);
```

# Kiểu chuỗi trong Dart

- **Đặc điểm:** Tập hợp các ký tự UTF-16
- **Khai báo:** Dùng dấu nháy đơn hoặc kép
- **Tính năng:** String interpolation (\$)

```
1 void main() {  
2     // Khai bao chuoi  
3     var s1 = 'Dau nhay don';  
4     var s2 = "Dau nhay kep";  
5     // String interpolation  
6     var name = 'Tung';  
7     print('Xin chao, $name!');  
8     // Chuoi nhieu dong  
9     var s3 = '''  
10    Day la chuoi  
11    nhieu dong  
12    ''';  
13    // Chuoi tho  
14    var s4 = r'Chuoi tho \n khong xu ly';  
15 }
```

# Kiểu Boolean trong Dart

- **Giá trị:** true hoặc false
- **Đặc điểm:** Không cho phép giá trị null
- **Sử dụng:** Trong các điều kiện, phép so sánh

```
1 void main() {  
2     bool isTrue = true;  
3     bool isFalse = false;  
4     // Kiểm tra điều kiện  
5     var fullName = '';  
6     assert(fullName.isEmpty);  
7     var hitPoints = 0;  
8     assert(hitPoints == 0);  
9     var unicorn;  
10    assert(unicorn == null);  
11 }
```

# Kiểu List trong Dart

- **Đặc điểm:** Mảng các phần tử có thứ tự
- **Index:** Bắt đầu từ 0
- **Tính năng:** Có thể thay đổi kích thước

```
1 void main() {  
2     // Khai báo list  
3     var list = [1, 2, 3];  
4  
5     // Truy cập phần tử  
6     assert(list.length == 3);  
7     assert(list[1] == 2);  
8  
9     // List hằng số  
10    var constantList = const [1, 2, 3];  
11  
12    // List với trailing comma  
13    var vehicles = [  
14        'Car',  
15        'Boat',
```



# Kiểu Set trong Dart

- **Đặc điểm:** Tập hợp các phần tử không trùng lặp
- **Tính năng:** Tự động loại bỏ phần tử trùng

```
1 void main() {  
2     // Khai bao Set  
3     var set = {1, 2, 3};  
4  
5     // Them phan tu  
6     set.add(4);  
7     set.add(1); // Khong them duoc vi trung  
8  
9     print(set); // {1, 2, 3, 4}  
10  
11    // Kiem tra phan tu  
12    assert(set.contains(2));  
13 }
```

# Kiểu Map trong Dart

- **Đặc điểm:** Tập hợp các cặp key-value
- **Key:** Phải là duy nhất
- **Value:** Có thể trùng lặp

```
1 void main() {  
2     // Khai báo Map  
3     var map = {  
4         'name': 'Tung',  
5         'age': 26  
6     };  
7  
8     // Thêm cặp key-value  
9     map['language'] = 'Tieng Viet';  
10  
11    // Truy cập value  
12    assert(map['name'] == 'Tung');  
13  
14    // Kiểm tra key  
15    assert(map.containsKey('age'));
```



# Runes trong Dart

- **Đặc điểm:** Chuỗi Unicode 32-bit
- **Cú pháp:** `\uXXXX` hoặc `\u{XX}`
- **Ứng dụng:** Xử lý emoji và ký tự đặc biệt

# Records trong Dart

- **Đặc điểm:**

- Kiểu dữ liệu tổng hợp
- Không thể thay đổi (immutable)
- Ẩn danh (anonymous)
- Kích thước cố định
- Không đồng nhất (heterogeneous)

```
1 // Tao record co ban
2 var record = ('first', a: 2, b: true, 'last');
```

# Cách sử dụng Records

```
1 void main() {  
2     // Record voi truong vi tri  
3     var point = (123, 456);  
4  
5     // Record voi truong co ten  
6     var person = (  
7         name: 'Bob',  
8         age: 25,  
9         city: 'Ha Noi'  
10    );  
11  
12    // Truy cap du lieu  
13    print(point.$1);           // 123  
14    print(person.name);       // Bob  
15 }
```

# Ứng dụng của Records

```
1 // Hàm trả về nhiều giá trị
2 (String, int) getUserInfo() {
3     return ('Bob', 25);
4 }
5
6 void main() {
7     // Sử dụng trong List
8     var list = [
9         (1, 'one'),
10        (2, 'two'),
11        (3, 'three'),
12    ];
13
14    // Phân ra thành phần
15    var (name, age) = getUserInfo();
16    print('$name is $age years old');
17 }
```

- **Đặc tính:**

- Giá trị không thể thay đổi sau khi tạo
- Có thể lồng nhau
- Có thể truyền vào/ra khỏi hàm
- Có thể lưu trữ trong Lists, Maps, Sets

- **Truy cập dữ liệu:**

- Trường vị trí: \$1, \$2, ...
- Trường có tên: .tenTruong

- **Câu lệnh điều khiển:**

- break
- continue

- **Các loại vòng lặp:**

- for
- for-in
- while
- do-while



# Câu lệnh if

```
1 void main() {  
2     bool isRaining = false;  
3     bool isSnowing = true;  
4  
5     if (isRaining) {  
6         print('Mang theo ao mua.');7     } else if (isSnowing) {  
8         print('Mac ao khoac.');9     } else {  
10        print('Ha mui xe xuong.');11    }  
12 }
```

# If-case và Switch-case

```
1 void main() {  
2     // If-case  
3     var pair = [3, 4];  
4     if (pair case [int x, int y]) {  
5         print('Cap toa do: $x, $y');  
6     }  
7  
8     // Switch-case  
9     var command = 'OPEN';  
10    switch (command) {  
11        case 'CLOSED':  
12            print('Dong cua.');  
13            break;  
14        case 'OPEN':  
15            print('Mo cua.');  
16            break;  
17        default:  
18            print('Lenh khong xac dinh.');  
19    }
```

# Vòng lặp for

```
1 void main() {  
2     // Vòng lặp for thông thường  
3     for (var i = 1; i <= 5; i++) {  
4         print(i);  
5     }  
6  
7     // Vòng lặp for-in  
8     var names = ['An', 'Binh', 'Chi'];  
9     for (var name in names) {  
10        print(name);  
11    }  
12  
13    // For với tập hợp  
14    var numbers = [1, 2, 3];  
15    numbers.forEach((number) => print(number));  
16 }
```

# While và Do-while

```
1 void main() {  
2     // Vòng lặp while  
3     var i = 1;  
4     while (i <= 5) {  
5         print(i);  
6         i++;  
7     }  
8  
9     // Vòng lặp do-while  
10    var j = 1;  
11    do {  
12        print(j);  
13        j++;  
14    } while (j <= 5);  
15 }
```

# Break và Continue

```
1 void main() {  
2     // Su dung break  
3     for (var i = 1; i <= 5; i++) {  
4         if (i == 3) {  
5             break;  
6         }  
7         print(i); // In: 1, 2  
8     }  
9  
10    // Su dung continue  
11    for (var i = 1; i <= 5; i++) {  
12        if (i == 3) {  
13            continue;  
14        }  
15        print(i); // In: 1, 2, 4, 5  
16    }  
17 }
```

- **Định nghĩa:**

- Cho phép tái sử dụng code với nhiều kiểu dữ liệu khác nhau
- Tăng tính linh hoạt và an toàn kiểu dữ liệu
- Giảm thiểu việc lặp lại code

- **Cú pháp:**

- Sử dụng `<T>` để định nghĩa kiểu generic
- T là tên thông thường cho kiểu generic

# Generic Collections

```
1 void main() {  
2     // List voi generics  
3     List<String> names = ['Bob', 'Alice'];  
4     List<int> numbers = [1, 2, 3];  
5  
6     // Map voi generics  
7     Map<String, int> scores = {  
8         'Bob': 90,  
9         'Alice': 95  
10    };  
11  
12    // Set voi generics  
13    Set<int> numberSet = {1, 2, 3};  
14 }
```

# Generic Classes

```
1 // Lop generic
2 class Box<T> {
3     T value;
4
5     Box(this.value);
6
7     T getValue() {
8         return value;
9     }
10 }
11
12 void main() {
13     // Su dung voi String
14     var stringBox = Box<String>('Hello');
15
16     // Su dung voi int
17     var numberBox = Box<int>(42);
18
19     print(stringBox.getValue()); // Hello
```



# Generic Methods

```
1 // Phương thức generic
2 T first<T>(List<T> list) {
3     return list[0];
4 }
5
6 // Sử dụng với nhiều kiểu
7 void printFirst<T>(List<T> list) {
8     print('Phan tu dau tien: ${list[0]}');
9 }
10
11 void main() {
12     var numbers = [1, 2, 3];
13     var strings = ['a', 'b', 'c'];
14
15     print(first(numbers)); // 1
16     print(first(strings)); // a
17
18     printFirst(numbers); // Phan tu dau tien:
```

1

# Giới hạn kiểu Generic

```
1 // Định nghĩa interface
2 abstract class Comparable {
3     int compareTo(other);
4 }
5
6 // Generic với giới hạn kiểu
7 class SortedList<T extends Comparable> {
8     List<T> items = [];
9
10    void add(T item) {
11        items.add(item);
12        items.sort((a, b) => a.compareTo(b));
13    }
14 }
15
16 // Sử dụng
17 class Person implements Comparable {
18     String name;
19     Person(this.name);
```



# Lợi ích của Generics

- **Tái sử dụng code**
  - Viết code một lần, sử dụng nhiều kiểu
  - Giảm số lượng code trùng lặp
- **An toàn kiểu**
  - Phát hiện lỗi tại thời điểm biên dịch
  - Tránh lỗi runtime
- **Hiệu suất**
  - Không cần ép kiểu thủ công
  - Code rõ ràng và dễ bảo trì

# Typedefs trong Dart

- **Định nghĩa:**

- Tạo bí danh (alias) cho kiểu dữ liệu
- Giúp code rõ ràng và dễ đọc
- Đặc biệt hữu ích với kiểu dữ liệu phức tạp

```
1 // Typedef cơ bản
2 typedef IntList = List<int>;
3
4 void main() {
5     IntList numbers = [1, 2, 3];
6     print(numbers); // [1, 2, 3]
7 }
```

# Typedefs với Generic

```
1 // Typedef voi tham so kieu
2 typedef ListMapper<X> = Map<X, List<X>>;
3
4 void main() {
5     // Cach viet dai
6     Map<String, List<String>> m1 = {};
7
8     // Ngan gon hon voi typedef
9     ListMapper<String> m2 = {};
10
11    // Ca hai deu tuong duong
12    print(m1.runtimeType == m2.runtimeType); //
13        true
14 }
```

# Typedefs cho Hàm

```
1 // Typedef cho ham
2 typedef Compare<T> = int Function(T a, T b);
3
4 // Ham so sanh
5 int sort(int a, int b) => a - b;
6
7 void main() {
8     // Kiem tra kieu
9     assert(sort is Compare<int>);
10
11     // Su dung typedef
12     Compare<int> comparator = sort;
13     print(comparator(5, 3)); // 2
14 }
```

# Ứng dụng của Typedefs

```
1 // Typedef cho callback
2 typedef Callback = void Function(String
   message);
3
4 class Notifier {
5     final Callback onNotify;
6
7     Notifier(this.onNotify);
8
9     void notify(String msg) {
10         onNotify(msg);
11     }
12 }
13
14 void main() {
15     final notifier = Notifier((msg) {
16         print('Thông báo: $msg');
17     });
18 }
```

# Lưu ý khi sử dụng Typedefs

- **Khi nên dùng:**

- Kiểu dữ liệu phức tạp, dài
- Kiểu dữ liệu được tái sử dụng nhiều
- Các signature của callback function

- **Thực hành tốt:**

- Đặt tên rõ ràng, có ý nghĩa
- Sử dụng cho cả generic types
- Tránh lạm dụng với kiểu đơn giản

- **Cập nhật:**

- Từ Dart 2.13: Hỗ trợ nhiều loại dữ liệu
- Trước đó: Chỉ dùng cho function types