Lộ trình Flutter điều chỉnh cho Embedded System

# Giai đoạn 1: Làm quen với Flutter cơ bản

## Học Dart (2 tuần):

- Hiểu cú pháp cơ bản: biến, hàm, lớp, vòng lặp, điều kiện.

- Học về các khái niệm OOP trong Dart.

## Học Flutter cơ bản (1 tháng):

- Giao diện người dùng:

+ Tìm hiểu về widget (Stateless, Stateful).

+ Xây dựng các màn hình giao diện cơ bản.

- Điều hướng (Navigation):

+ Học cách chuyển đổi giữa các màn hình.

- Quản lý trạng thái:

+ Tìm hiểu các phương pháp như setState, Provider, hoặc Riverpod.

## Thực hành:

- Tạo một ứng dụng Flutter cơ bản, ví dụ: Ứng dụng ghi chú hoặc danh sách công việc.

# Giai đoạn 2: Học Flutter nâng cao và tích hợp nhúng

## Kết nối ứng dụng với Internet (1 tháng):

- Học cách làm việc với API RESTful:

+ Gửi và nhận dữ liệu từ server.

+ Xử lý JSON.

- Tìm hiểu giao thức WebSocket để truyền dữ liệu thời gian thực.

## Làm việc với cơ sở dữ liệu:

- Sử dụng Firebase hoặc SQLite để lưu trữ dữ liệu.

## Tích hợp Flutter với Embedded Systems:

- Bluetooth:

+ Học cách sử dụng các plugin Flutter như flutter\_blue để kết nối với thiết bị nhúng qua Bluetooth.

- Wi-Fi:

+ Dùng giao thức HTTP hoặc MQTT để giao tiếp với thiết bị nhúng (ESP32, Raspberry Pi).

- MQTT:

+ Tìm hiểu cách sử dụng Flutter để giao tiếp với server MQTT (Adafruit IO, Mosquitto).

## Thực hành:

- Tạo ứng dụng điều khiển thiết bị thông qua Bluetooth hoặc Wi-Fi.

- Xây dựng dashboard hiển thị dữ liệu cảm biến.

# Giai đoạn 3: Phát triển ứng dụng nhúng toàn diện

## Xây dựng Plugin tùy chỉnh (1 tháng):

- Viết plugin Flutter bằng C/C++ để giao tiếp trực tiếp với phần cứng qua USB hoặc Serial.

- Học cách sử dụng MethodChannel để tương tác giữa Flutter và các thư viện native.

## Tối ưu hóa ứng dụng:

- Sử dụng thiết kế Material Design để làm ứng dụng trực quan hơn.

- Học cách tối ưu hiệu năng ứng dụng Flutter (load dữ liệu nhanh, UI mượt mà).

## Thực hành dự án lớn:

- Dự án IoT hoàn chỉnh:

+ Kết nối ESP32 hoặc Raspberry Pi với ứng dụng Flutter.

+ Điều khiển thiết bị và nhận dữ liệu cảm biến (như nhiệt độ, độ ẩm).

- Hệ thống giám sát:

+ Tạo ứng dụng Flutter hiển thị trạng thái thiết bị (ON/OFF, dữ liệu cảm biến) và điều khiển thiết bị từ xa.

# Dart Programming

## Data Type:Quan tâm tới 4 thằng: var, final, const, Object

### **var**

Khai báo biến thay đổi đượ giá trị, suy đoán được giá trị bạn nhập vào. Xem ví dụ phía dưới.

Ví dụ 1:

|  |
| --- |
| void main(){    var myVariable = '1'; // String    print(myVariable); // Output: 1    print(myVariable.runtimeType); // Output: String    var myVariable\_1 = 1; // int    print(myVariable\_1); // Output: 1    print(myVariable\_1.runtimeType); // Output: int    var myVariable\_2 = 1.2; // Double    print(myVariable\_2); // Output: 1.2    print(myVariable\_2.runtimeType); // Output: Double    var myVariable\_3 = true; // bool    print(myVariable\_3); // Output: true    print(myVariable\_3.runtimeType); // Output: bool  } |

Table 1: Ví dụ cho việc dùng var thay vì dùng cách như C programming

Ví dụ 2:

|  |
| --- |
| void main() {    var x = 10;   // Dart suy luận kiểu của x là int    print(x);     // Output: 10      x = 20;       // Có thể thay đổi giá trị    print(x);     // Output: 20  } |

Table 2: ví dụ về thay đổi giá trị

### **final**

Dùng để khai báo biến chỉ có thể gán giá trị **một lần duy nhất.** (gần giống **const**)

Giá trị được gán ngay tại thời điểm chạy (**runtime**).

Tự động hiểu kiểu dữ liệu là gì nếu không chỉ rõ.

Ví dụ 3:

|  |
| --- |
| void main() {    final y = 30;  // Dart suy luận kiểu của y là int    print(y);      // Output: 30    // y = 40;     // Lỗi: Không thể gán giá trị mới cho final variable  } |

Table 3

### **const**

Khai báo biến có giá trị cố định và không đổi tại thời điểm biên dịch (**compile-time**).

Không phụ thuộc **runtime**.

Các giá trị được lưu trong bộ nhớ bất biến (**immutable memory**).

Sử dụng khi muốn giá trị không đổi và biết giá trị tại thời điểm biên dịch

Ví dụ:

|  |
| --- |
| void main() {    const pi = 3.14;  // Biến bất biến và giá trị được xác định tại compile-time    print(pi);        // Output: 3.14    // const currentTime = DateTime.now(); // Lỗi: const không cho phép giá trị runtime  } |

Table 4

### **Object**

là lớp gốc (**base class**) của tất cả các kiểu dữ liệu trong Dart, kể cả các kiểu cơ bản (**int**, **String**) và các kiểu do người dùng định nghĩa.

Một biến kiểu Object có thể chứa giá trị của bất kỳ kiểu dữ liệu nào.

Tuy nhiên, khi làm việc với Object, bạn cần kiểm tra kiểu dữ liệu (type-checking) hoặc ép kiểu (casting) để sử dụng các phương thức đặc trưng.

Sử dụng khi cần lưu trữ hoặc xử lý dữ liệu mà kiểu của nó có thể thay đổi hoặc không biết trước.

Ví dụ:

|  |
| --- |
| void main() {    Object obj = "Hello";  // Biến obj chứa giá trị kiểu String    print(obj);            // Output: Hello    obj = 42;              // Biến obj chứa giá trị kiểu int    print(obj);            // Output: 42    if (obj is int) {      print(obj + 10);     // Ép kiểu thành int và sử dụng    }  } |

Table 5

### **Phân biệt var, final, const, Object**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Đặc điểm | var | final | const | Object |
| **Thay đổi giá trị** | **✅** | Không thể thay đổi giá trị sau khi gán. | ❌ | Có thể gán lại giá trị mới (mutable). |
| **Suy luận kiểu** | ✅ | ✅ | ✅ | ❌  luôn là Object. |
| **Thời điểm xác định giá trị** | Lúc runtime. | Lúc runtime. | Lúc compile-time. | Lúc runtime. |
| **Kiểm tra kiểu** | Không cần kiểm tra kiểu (an toàn kiểu). | Không cần kiểm tra kiểu (an toàn kiểu). | Không cần kiểm tra kiểu (an toàn kiểu). | Cần kiểm tra hoặc ép kiểu khi sử dụng. |
| **Khi sử dụng** | Khi cần biến có thể thay đổi. | Khi cần biến có giá trị cố định tại runtime. | Khi cần biến cố định tại compile-time. | Khi cần lưu trữ dữ liệu không xác định kiểu. |

Table 6

### **Runtime và Compile Time**

|  |  |
| --- | --- |
| Run Time | Compile Time |
| Khoản thời gian một chương trình đang chạy và thời gian này thường xảy ra sau **Compile Time** | Khoản thời gian programming code chuyển thành machine code |
| Dùng để chạy code | Dùng để kiểm tra syntax và semantics code |
| Phát hiện lỗi sau khi thực thi chương trình | Phát hiện lỗi mà không cần thực thi chương trình |
| Quay lại code để sửa lỗi | Fix lỗi tại chỗ |

### **Khi nào nên dùng?**

|  |
| --- |
| **var**: Khai báo biến có thể thay đổi giá trị và để Dart suy luận kiểu tự động.  **final**: Khai báo biến chỉ gán giá trị một lần nhưng giá trị đó có thể được xác định tại runtime.  **const**: Khai báo biến cố định mà giá trị phải được biết tại compile-time.  **Object**: Khi cần lưu trữ dữ liệu không biết trước kiểu cụ thể hoặc có thể chứa bất kỳ loại dữ liệu nào. |

Table 7

### **List**

#### Syntax:

|  |
| --- |
| List<data\_type> variable\_name = [element\_1, element\_2, ...]; |

#### Ví dụ:

|  |
| --- |
| void main() {    var list = [1, 2, 3, 'String', true];    print(list.runtimeType); // List<Object>    List<int> ok= [1, 2, 3];    print(ok.runtimeType);  // List<int>    List<dynamic> okela = [int, bool];    print(okela.runtimeType); // List<dynamic>  } |

Table 8

### **Map**

#### Syntax:

|  |
| --- |
| Map<data\_type\_key, data\_type\_val> variable\_name = {    key\_1 : value\_1,    key\_2 : value\_2,    ...  }; |

#### Ví dụ:

|  |
| --- |
| void main() {    Map<String, int> persons\_1 = {      'John' : 21,      'Nick' : 23,      'Khải' : 24    };    print(persons\_1.runtimeType); // \_Map<String, int>  } |

Table 9

|  |
| --- |
| void main() {    var persons\_2 = {      'John' : 21,      'Nick' : 23,      'Khải' : 24,      0      : 'Không'    };    print(persons\_2.runtimeType); // \_Map<Object, Object>  } |

Table 10

|  |
| --- |
| void main() {    Map<dynamic, dynamic> persons\_3 = {      'John' : 21,      'Nick' : 23,      'Khải' : 24,      0      : 'Không'    };    print(persons\_3.runtimeType); // \_Map<dynamic, dynamic>  } |

Table 11

## Conditions

### **If condition**

#### Syntax:

|  |
| --- |
| if(condition) {      Statement 1;      Statement 2;         .         .      Statement n;  } |

#### Ví dụ:

|  |
| --- |
| void main()  {      var age = 20;        if(age >= 18){        print("You are voter.");      }  } |

### **If-else condition**

#### Syntax:

|  |
| --- |
| if(condition){  statements;  }else{  statements;  } |

#### Ví dụ:

|  |
| --- |
| void main(){          int age = 12;          if(age >= 18){              print("You are voter.");          }else{              print("You are not voter.");          }  } |

### **If-else-if condition**

#### Syntax:

|  |
| --- |
| if(condition1){  statements1;  }else if(condition2){  statements2;  }else if(condition3){  statements3;  }  .  .  .  else(conditionN){  statementsN;  } |

#### Ví dụ:

|  |
| --- |
| **void main(){**  **int age = 20;**  **String vote = voteFunction(age);**  **}**  **voteFunction(int age){**  **if (age >= 18 && age < 100){**  **return 'You can vote';**  **} else if (age >= 100){**  **return 'Are you sure that you still alive ! ! !';**  **} else if (age <= 0){**  **return 'You are not born';**  **} else{**  **return 'You can not vote';**  **}**  **}** |

### **Ternary condition**

#### Syntax:

|  |
| --- |
| condition ? exprIfTrue : exprIfFalse |

#### Ví dụ:

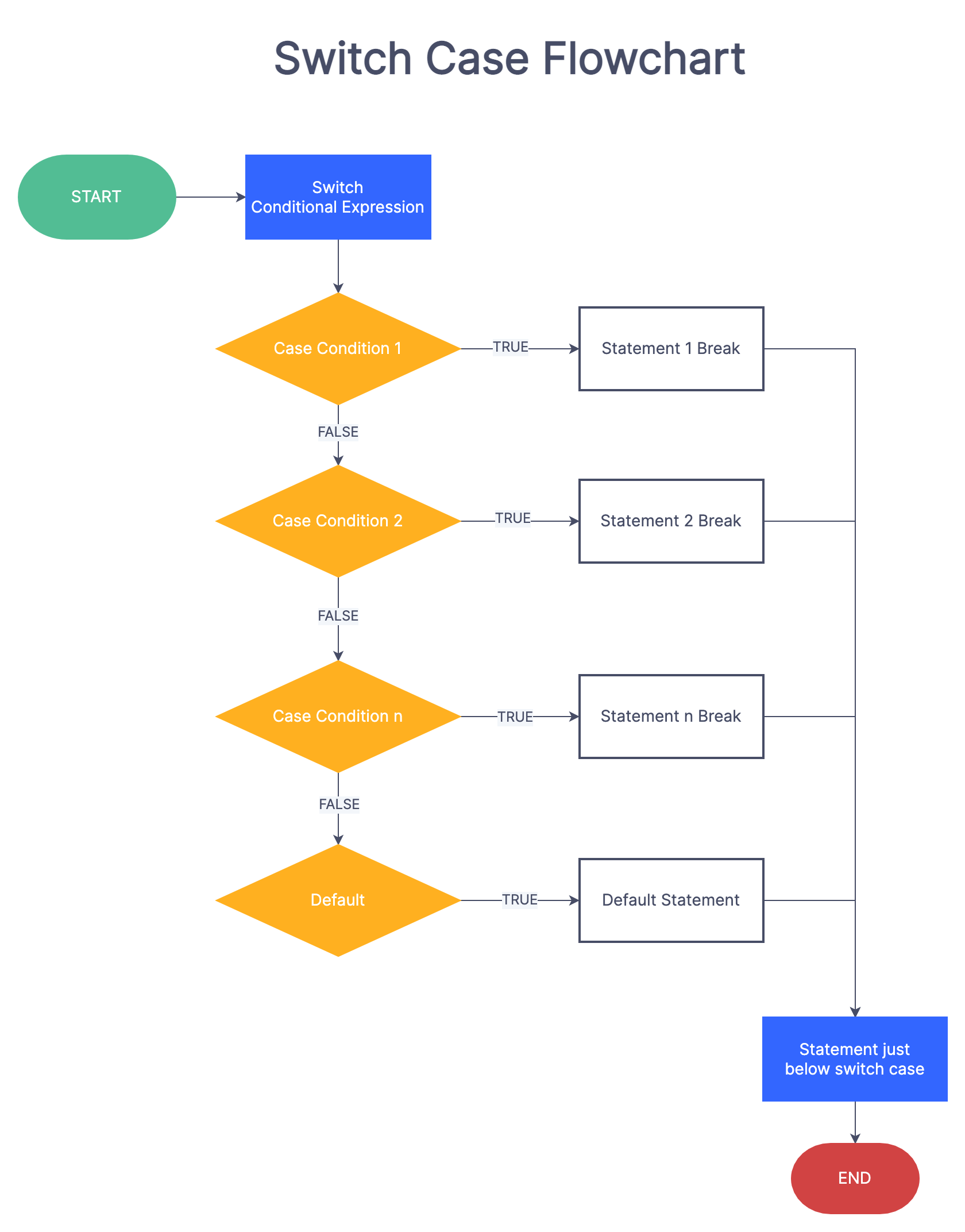
|  |
| --- |
| **void main(){**  **int age = 20;**  **String vote = voteFunction(age);**  **}**  **voteFunction(int age){**  **if (age >= 18 && age < 100){**  **return 'You can vote';**  **} else if (age >= 100){**  **return 'Are you sure that you still alive ! ! !';**  **} else if (age <= 0){**  **return 'You are not born';**  **} else{**  **return 'You can not vote';**  **}**  **}** |

If-else-if condition

|  |
| --- |
| void main(){    int age = -4;    String vote = voteFunction(age);    print(vote);  }  voteFunction(int age){    var res = (age >= 18 && age < 100) ? 'You can vote'    : (age >= 100) ? 'Are you sure that you still alive ! ! !'    : (age <= 0) ? 'You are not born'    : 'You can not vote';    return res;  } |

Ternary condition

### **Switch case condition**



#### Syntax:

|  |
| --- |
| switch ( expression ) {     case value1: {        // Body of value1     } break;     case value2: {        //Body of value2     } break;     .     .     .     default: {        //Body of default case     } break;  } |

#### Ví dụ:

|  |
| --- |
| void main(){    int choice = 1;    print('Your choice is:');    voteFunction(choice);  }  voteFunction(int choice){    switch (choice){      case 1: {        print('Say hello ! ! !');      } break;      case 2: {        print('Say goodbye ! ! !');      } break;      default: {        print('Invalid Choice ! ! !');      } break;    }  } |

## Functions

### **Syntax:**

|  |
| --- |
| returntype functionName(parameter1,parameter2, ...){    // function body  } |

### **Ví dụ:**

|  |
| --- |
| // writing function outside main function.  void printName(){    print("My name is Raj Sharma. I am from function.");  }  // this is our main function.  void main(){    printName();  } |

Ví dụ 1

|  |
| --- |
| String greeting(String name){    return ('Hello' + name);  }  void main(){    print(greeting('Ta ke'));  } |

Ví dụ 2

|  |
| --- |
| void main(){    no\_return\_func();    print(return\_func());  }  void no\_return\_func() => print("No return Function !");  String return\_func() => "Return Function !"; |

Ví dụ 3

## Class

### **Tạo một class:**

#### Các biến instance (hoặc gọi là Attribute Variables – biến thuộc tính trong class):

Tạo một class Person có 3 attribute variables như dưới đây:

|  |
| --- |
| class Person{  String ? name; // khỏi tạo biến name với giá trị null  String ? gender; // khỏi tạo biến name với giá trị null  int ? age = 0; // khỏi tạo biến age với giá trị 0 } |

#### Các method:

Tạo hai method là walking và talking:

|  |
| --- |
| class Person{  String ? name; // khỏi tạo biến name với giá trị null  String ? gender; // khỏi tạo biến name với giá trị null  int ? age = 0; // khỏi tạo biến age với giá trị 0   walking() => print("$name is walking"); // void function  talking() => "$name is talking"; // return function (String) } |

#### Ví dụ cụ thể:

|  |
| --- |
| class Person{  String ? name; // khỏi tạo biến name với giá trị null  String ? gender; // khỏi tạo biến name với giá trị null  int ? age = 0; // khỏi tạo biến age với giá trị 0   walking() => print("$name is walking"); // void function  talking() => "$name is talking"; // return function (String) }  void main(){  /\* ================================= \*/  Person p1 = Person();   p1.name = "A";  p1.age = 20;  p1.gender = "Male";   p1.walking(); // Output: A is walking  print(p1.talking()); // Output: A is talking   /\* ================================= \*/  Person p2 = Person();   p2.name = "B";  p2.age = 23;  p2.gender = "Female";   p2.walking(); // Output: B is walking  print(p2.talking()); // Output: B is talking } |

### **Phạm vi truy xuất**

Xem ví dụ dưới đây:

|  |
| --- |
| class User{  // *TODO: Attribute declaration (Khỏi tạo các thuộc tính của class)* // *todo: Public variables* int id;  String name;  // *todo: Private variables* String \_password;   // *TODO: Constructor (hàm khởi tạo)* User(this.id, this.name, this.\_password);   // *TODO: Getter and Setter* // No Getter => can't print  // No Setter => can't change  String get password => \_password;   set password(String value) {  \_password = value;  }   // *TODO: Display info using toString* @override  String toString() {  // *TODO: implement toString* // Check admin  return (name == 'admin') ?  '$id - $name - $\_password' : '$id - $name';  } } |

Class User chứa 3 thuộc tính (2 public: name, id; 1 private: password), một constructor function, getter and setter function, và một method.

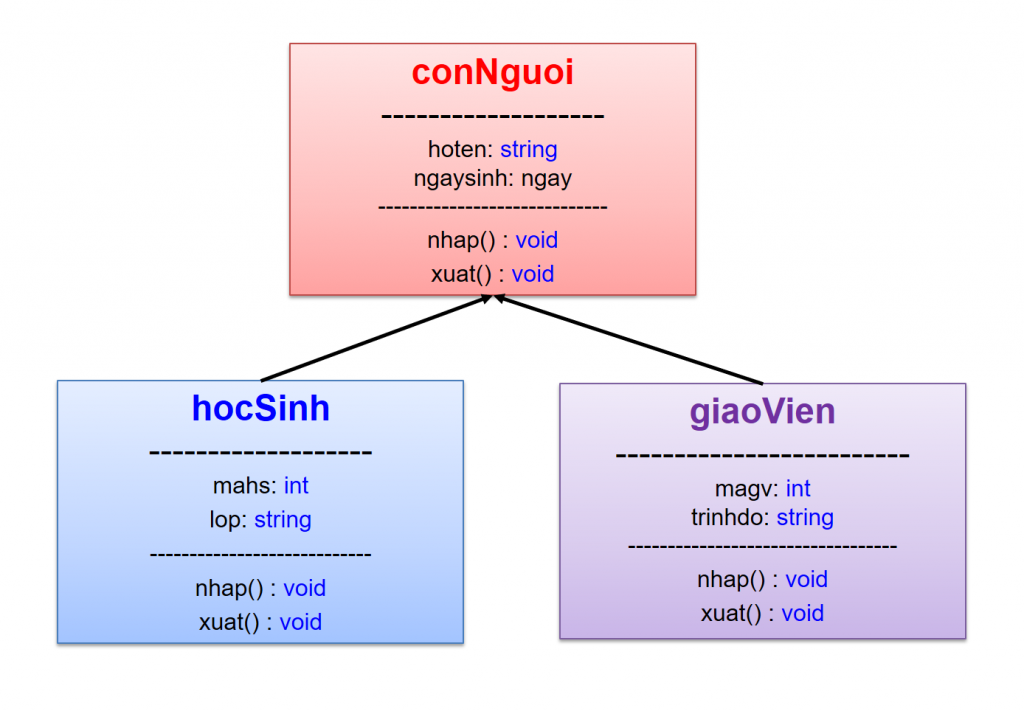
Hàm main():

|  |
| --- |
| void main() {  User user\_0 = User(0, 'admin', '123');  // user\_0.name = 'HTK';  user\_0.id = 1;  user\_0.password = '1223'; // Use Getter to change password.   print(user\_0.toString());   print(user\_0.password); // Use Setter to print password. } |

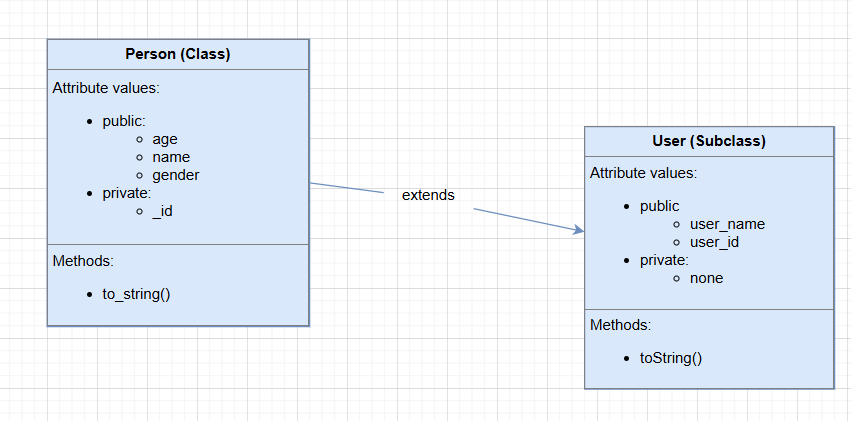
Chạy và xem kết quả: 1. Giữ nguyên, 2. Xoá hàm set, 3. Xoá hàm get => hiểu hơn =))

### **Tính kế thừa (Inheritance, trong Java gọi là extends)**

#### Hình ảnh minh hoạ:



#### Ví dụ đơn giản:



#### Code:

* Person Class (Parent / Class):

|  |
| --- |
| class Person{  // *TODO: Attribute declaration (Khỏi tạo các thuộc tính của class)* // *todo: Public variables* int age;  String name;  String gender;  // *todo: Private variables* String \_id;   // *TODO: Constructor (hàm khởi tạo)* Person(this.age, this.name, this.gender, this.\_id);   // *TODO: Getter and Setter* // No Getter => can't print  // No Setter => can't change  String get id => \_id;   set id(String value) {  \_id = value;  }   // *TODO: Display info using toString* @override  String to\_string() {  // *TODO: implement toString* return '$name - $age - $gender - $\_id';  } } |

* User Class (Child / Subclass):

|  |
| --- |
| import 'package:beginer\_tutorial/model/person.dart';  class User extends Person{  String user\_id;  String user\_name;   User(this.user\_id, this.user\_name) : super(18, 'Kim', 'female', '1234321');   @override  String toString() {  // *TODO: implement toString* return '$user\_id - $user\_name - ${super.to\_string()}';  } } |

* main() function:

|  |
| --- |
| void main() {  User user\_0 = User('KimCang\_1718', 'Kim LA');  print(user\_0.toString());  print(user\_0.to\_string()); } |