Ngôn ngữ Dart căn bản (phần 2)

...

Xây dựng hàm

Hàm trong Dart

- Hàm là một khối lệnh thực hiện một tác vụ gì đó, khối lệnh này được dùng nhiều lần nên gom chúng thành một hàm
- Trong Dart, mọi thứ là đối tượng nên hàm cũng là một đối tượng

Khai báo hàm

Khai báo hàm gồm 3 phần: tham số truyền vào, kiểu trả về và định nghĩa hàm:

- Kiểu trả về của hàm (có thể khuyết) được viết trước tên hàm
- Tham số truyền vào (có thể khuyết) được khai báo trong ngoặc tròn (liền sau tên hàm)
- Định nghĩa hàm nằm trong cặp dấu {}, mọi thứ hàm xử lý được viết ở đây

Ví dụ:

```
//double là kiểu hàm trả về
// a, b, c là các tham số truyền vào
double tinhtong(var a, double b, double c) {
  return a + b + c; // giá trị trả về phải trùng với kiểu trả về
}
//GỌI HÀM
var x = tinhtong(1,2,3);
print(x); //6.0
```

Giá trị tham số mặc định của hàm

Nếu bạn muốn tham số có giá trị mặc định, nghĩa là khi gọi hàm mà thiếu giá trị cho tham số đó, thì nó sẽ nhận mặc định:

```
double tinhtong(var a, {double b:1}) {
  return a + b;
}
var sum1 = tinhtong(1, b:10); // truyền đầy đủ tham số
print(sum1); //11.0

var v1 = tinhtong(1); //b được lấy mặc định là 1
print(v1); //2.0
```

Hàm với tham số tuỳ chọn

Các tham số tuỳ chọn là những tham số có thể sử dụng hoặc không

```
double tinhtong(var a, [double b, double c]) {
   var tong = a;
   if (b != null)
    tong += a;
   tong += (c!=null) ?  c: 0;
}
print(tinhtong(1));        //1.0
print(tinhtong(1,2));      //3.0;
print(tinhtong(1,2,3));    //6.0;
```

Hàm với ký hiệu =>

Với những hàm chỉ có một biểu thức trả về luôn, thì có thể có cách viết ngắn gọn bằng ký hiệu mũi tên

```
double tinhtong(var a, var b) {
    return a + b;
  }

  // Có thể viết lại thành
  double tinhtong(var a, var b) => a + b;
```

Hàm ẩn danh - lambda/closure

- Hầu hết khai báo hàm là có tên hàm, tuy nhiên trong nhiều ngữ cảnh khai báo hàm không dùng đến tên, hàm đó gọi là hàm ẩn danh (hay còn gọi là lambda hoặc closure)
- Tạo ra hàm ẩn danh thì cú pháp như khai báo hàm, nhưng không có tên và không có kiểu trả về

Ví dụ hàm ẩn danh

```
(var a, var b) {
    return a + b;
};
//Có thể dùng ký hiệu mũi tên  () => {}
(var a, var b) => {
    return a + b;
}
//Nếu chỉ có 1 biểu thức trả về như trên có thể viết gọn hơn
(var a, var b) => return a + b;
```

Sử dụng hàm ẩn danh

Để sử dụng hàm ẩn danh thì ra lệnh cho nó gọi luôn hoặc gán nó vào một biến rồi dùng biến gọi hàm.

```
//Khai báo xong chạy luôn
  var x = (var a, var b) {
    return a + b;
  }(5, 6);
  print(x);
//Gán hàm ẩn danh vào biến ham, rồi dùng nó chạy
  var ham = (var a, var b) {
    return a + b;
  };
  print(ham(10, 11));
```

Sử dụng lambda làm tham số

Hàm ẩn danh có thể dùng làm tham số (callback) trong các hàm khác, thậm chí khai báo luôn một ẩn danh ở tham số hàm.

```
f1(var a, var b, var printmessage) {
     var c = a + b;
     printmessage(c);
}

f1(1, 2, (x) { print('Tổng là: $x');});          //Tổng là: 3
f1(1, 2, (z) => print('SUM = $z'));              //SUM = 3
```

Bài tập sử dụng hàm

Viết hàm:

1, Kiểm tra một số có phải số nguyên tố không.

2, Đếm các số nguyên tố nằm trong đoạn [0,100].

3, Tính tổng các số nguyên tố nằm trong đoạn [0,100].

Lớp trong Dart

Class trong Dart

- Class để tạo ra các đối tượng, với Dart mọi thứ kể cả số đều là đối tượng, các đối tượng đều kế thừa từ class Object
- Trong một class có thể có các thành phần:

❏ Các phương thức khởi tạo - *Hàm khởi tạo*
❏ Các biến lưu dữ liệu của đối tượng - *Thuộc tính*
❏ Các hàm (thành viên hàm) - *Phương thức*
❏ Các hàm đặc biệt thực hiện gán thuộc tính/truy cập thuộc tính - *hàm setter/getter*

Khai báo class - từ khoá class

```
class Product {
  //Khai báo các thuộc tính
  String manufacture = '';
  String name = '';
  var    price;
  int    quantity;
  //Khai báo hàm khởi tạo
  Product(var price, {int quantity:0}) {
  //this để gọi đến đối tượng trong class
    this.price    = price;
    this.quantity = quantity;
  }
...
```

```
...
  //Khai báo các phương thức
  calulateTotal() {
    return this.price * this.quantity;
  }
  showTotal() {
    var tong = this.calulateTotal();
    print("Tổng số tiền là: $tong");
  }
}
```

Sử dụng class

```
  // khởi tạo đối tượng từ class Product
var product = new Product(600, quantity: 1);
  // Sử dụng toán tử . để truy cập thuộc tính và phương thức 
của đối tượng
product.showTotal(); // kết quả: Tổng số tiền là: 600
product.quantity = 2; // truy cập để thay đổi giá trị thuộc 
tính của đối tượng
product.showTotal(); // kết quả: Tổng số tiền là: 1200
```

Hàm khởi tạo có tên

```
class Product {
// ...
Product.iphone(var price, {int quantity:0}) {
  this.price        = price;
  this.quantity     = quantity;
  this.manufacture = 'Apple';
}
// ...
}
```

```
// Sử dụng:
var product = Product.iphone(700, quantity: 2);
```

Phương thức tĩnh (static)

- Các phương thức trong lớp chỉ truy cập được trên một đối tượng cụ thể triển khai từ lớp, nhưng chúng ta có thể chỉ định cho phương thức đó trở thành tĩnh bằng từ khoá `static` thì hàm không cần đối tượng triển khai từ lớp mà vẫn gọi được thông qua tên lớp.

Ví dụ hàm tĩnh

```
class Product {
  // ...
  static showListStore() {
    print('Store 1 ...');
  }
  // ...
}
```

Chỉ cần tên lớp để gọi hàm static (hàm này thuộc về lớp chứ không thuộc về đối tượng triển khai từ lớp)

```
Product.showListStore();
```

Hàm setter/getter

- Chúng ta có thể xây dựng hàm đặc biệt mà chúng chỉ thực thi khi được gọi (setter gọi khi thực hiện gán, getter gọi khi truy cập)
- Sử dụng từ khoá get trước một hàm không có tham số thì hàm đó trở thành getter, sử dụng từ khoá set trước hàm 1 tham số thì đó là hàm setter
- Khi sử dụng setter/getter thì tên hàm phải giống nhau

Ví dụ hàm setter/getter

```
class Product {
  // ...
  //getter
  get nameProduct {
    return this.name;
  }
  //Hàm setter
  set nameProduct(name) {
    this.name = name;
...
```

```
...
switch (this.name) {
      case 'Iphone 6':
        this.manufacture = "Apple";
      break;
      case 'Galaxy S6':
        this.manufacture = 'Samsung';
      break;
      default: this.manufacture = '';
    }}
  // ...
}
```

Truy cập hàm setter/getter

Sau khi có hàm setter/getter thì truy cập giống như thuộc tính:

```
//Gọi đến hàm Setter - nameProduct
product.nameProduct = "Galaxy S6";
//Gọi đến Getter
var info = product.nameProduct;
```

Tính kế thừa trong class

- Từ một class đã có, chúng ta có thể định nghĩa lớp mới, lớp mới này sẽ kế thừa lớp cha (bao gồm các thuộc tính, phương thức)
- Để xây dựng một lớp mới kế thừa lớp đã có dùng tới từ khoá extends

Ví dụ kế thừa:

```
class Table extends Product {
  double length = 0;
  double width   = 0;
  Table(var giatien) : super(giatien, quantity:1) {
    //Mã khởi tạo tại lớp con, sau khi hàm tạo lớp cha chạy xong
    this.name = "Bàn Ăn";
  }
}
```

Khởi tạo tại lớp con và truy cập đến lớp cha

- Hàm khởi tạo ở lớp con, bắt buộc phải gọi một hàm khởi tạo nào đó của lớp cha. Để làm được điều đó sau hàm khởi tạo của lớp con chỉ rõ hàm tạo nào của lớp cha sẽ gọi sau dấu :
- Lớp con sẽ có những thuộc tính và phương thức kế thừa từ lớp cha, để truy cập ta sử dụng từ khoá this, lúc này this sẽ sử dụng phương thức định nghĩa lại
- Nếu muốn truy cập đến phiên bản định nghĩa bởi lớp cha sẽ dùng ký hiệu super thay cho this

Sử dụng lớp con

- Ở ví dụ trước, đoạn `super(giatien, quantity:1)` tương đương với hàm khởi tạo `Product(giatien, quantity:1)`
- Trở lại ví dụ lớp `Table`, khi tạo đối tượng từ lớp:

```
var table = new Table(500);
```

- Hàm thực thi nó đã gọi đến hàm tạo của lớp cha Product rồi đến các code của chính hàm tạo Table

Nạp chồng phương thức

- Tạo ra phiên bản mới của một phương thức đã có ở lớp cha, và từ đây đối tượng sẽ sử dụng phương thức mới được định nghĩa, để làm điều đó, ở lớp con tạo lại phương thức với chỉ thị `@override` - nạp chồng phương thức

Ví dụ nạp chồng phương thức

```
class Table extends Product {
  // ...
  @override
  showTotal() {
      print('Sản phẩm:' + this.name);
      //gọi đến phương thức ở lớp cha
      super.showTotal();
    }
 // ...
}
```

- Ở phiên bản lớp con, do có nhu cầu sử dụng lại phương thức của lớp cha nên nó gọi đến phương thức cũ bằng `super.showTotal()`
- Như vậy, các đối tượng triển khai từ lớp `Table` đã có một phiên bản riêng của phương thức showTable, mặc định nó sẽ gọi đến phương thức mới này (Kể cả các phương thức lớp cha cũng sẽ tự động gọi đến phương thức mới định nghĩa này)

```
var table = new Table(500);
  table.showToTal(); // Tổng tiền là: 500
```

Lớp trừu tượng

- Lớp trừu tượng là lớp không dùng trực tiếp để tạo ra đối tượng, nó chỉ sử dụng để lớp khác kế thừa. Phương thức trong lớp trừu tượng chỉ khai báo tên hàm, thì phương thức đó gọi là phương thức trừu tượng, lớp con kế thừa lớp trừu tượng bắt buộc phải định nghĩa hàm này
- Từ khoá khai báo lớp trừu tượng là abstract

Ví dụ lớp trừu tượng

```
abstract class A {
  //Khai báo các thuộc tính
  var name = 'My Abstract Class';
  //Khai báo phương thức trừu tượng (chỉ có tên)
  void displayInfomation();
}
```

Kế thừa lớp trừu tượng

```
// Class A không thể dùng tạo ra đối tượng, nhưng nó được lớp khác kế thừa
// Lớp con kế thừa bắt buộc phải định nghĩa nội dung cho phương thức trừu tượng bằng cách nạp chồng (@override).
// Ví dụ khai báo lớp B kế thừa lớp trừu tượng A
class B extends A {
  @override
  void displayInfomation() {
     print(this.name);
  }
}
```

Interface

- Interface - giao diện: là khái niệm quen thuộc trong các ngôn ngữ lập trình hướng đối tượng
- Các interface xác định một tập hợp các phương thức có sẵn trên 1 đối tượng
- Với Dart, interface được triển khai bởi lớp khác bằng từ khoá implements
- Khi một lớp được coi là một interface thì nó phải định nghĩa lại mọi phương thức, thuộc tính có trong interface
- Mục đích sử dụng interface là để đảm bảo các lớp có cùng giao diện sẽ có API giống nhau

Ví dụ interface

- Ví dụ xây dựng lớp C triển khai từ lớp B bằng implements. Vậy B sẽ là interface và trong C bắt buộc phải định nghĩa lại mọi thứ trong B

```
class C implements B {
  @override
  String name;

  @override
  void displayInfomation() {
      // ...
  }
}
```

Mixin

- Trong Dart thì Mixin là một lớp, nó không được sử dụng trực tiếp tạo ra đối tượng, một Mixin chứa các phương thức, thuộc tính dùng để gộp vào một lớp khác.
- Ví dụ ta tạo một Mixin là M, thì khi khai báo lớp C ở ví dụ trước muốn gộp những gì có ở M vào dùng từ khoá with mang ý nghĩa gộp code chứ không mang ý nghĩa kế thừa

Ví dụ Mixin

```
mixin M {
  var var1 = null;
  showSomething()
  {
    print('Print message
...');
  }
}
```

```
class C extends B with M {
  @override
  String name;

  @override
  void displayInfomation() {
    }
}
```

Một số lưu ý về Class

- Lớp Object: là cơ sở của Dart, mặc định mọi lớp, mọi hàm … kể cả lớp, hàm do chúng ta định nghĩa đều mở rộng từ lớp này. Như vậy mọi lớp đều có một số phương thức, thuộc tính chung là:

❖ hasdCode: thuộc tính chứa mã hash của đối tượng

❖ toString(): trả về chuỗi mô tả đối tượng

❖ ==: toán tử so sánh theo hashCode của hai đối tượng

Một số lưu ý về Class (tiếp)

- Khi có một chuỗi tác vụ trên đối tượng (gọi phương thức, thiết lập thuộc tính) thay vì phải viết đầy đủ đối tượng thì chỉ cần viết nó một lần, các tương tác tiếp theo thay bằng `..`

```
var table = new Table(1);
 table
 ..calulateTotal()      //Thay cho table.calulateTotal();
 ..length=100           //Thay cho table.length=100;
 ..name='Abc'
 ..quantity=100
 ..showTotal();
```

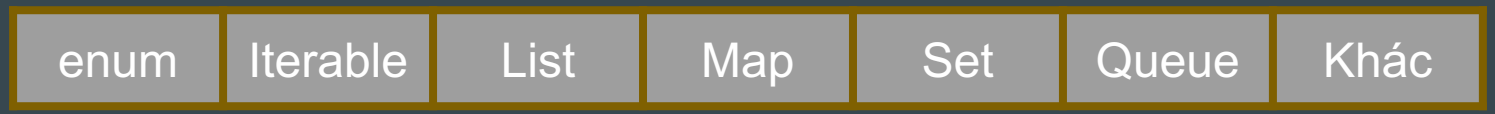
Bài tập về class

Xây dựng lớp Candidate (Thí sinh) gồm các thuộc tính: mã, tên, ngày tháng năm sinh, điểm thi Toán, Văn, Anh và các phương thức cần thiết.

Xây dựng lớp TestCandidate để kiểm tra lớp trên:

- Nhập vào n thí sinh (n do người dùng nhập)
- In ra thông tin về các thí sinh có tổng điểm lớn hơn 15

Cấu trúc dữ liệu



Cấu trúc dữ liệu liệt kê - enum

- enum biểu diễn một tập hợp cố định các hằng số.

```
enum UserGroup {guest, member, admin}
```

- enum tiện lợi để dùng với câu lệnh switch

```
var user_group = UserGroup.admin;
switch (user_group) {
    case UserGroup.admin:
      print('Quản trị hệ thống');
    break;
    case UserGroup.guest:
      print('Khách');
    break;
    default:
}
```

Cấu trúc dữ liệu - Iterable

- Iterable là một lớp generic biểu diễn tập dữ liệu mà có thể duyệt qua hết phần tử này đến phần tử khác
- Iterable được tạo ra, liên kết với một loại kiểu dữ liệu tập hợp khác như List, Map

```
var iterable = new Iterable.generate(100); //khởi tạo Iterable chứa 100 phần tử từ 0 - đến 99
for (var item in iterable) { // duyệt
    print(item);
}
// hoặc duyệt bằng forEach()
iterable.forEach((f) {
    print(f);
});
```

Tập hợp - Set

- Set là một tập hợp các phần từ, đảm bảo mỗi phần tử chỉ xuất hiện một lần
- Set duyệt phần tử giống với Iterable

```
  var s = Set(); // khởi tạo Set
  s.add('a'); // thêm phần tử
  s.remove('a'); // xoá phần từ
  s.contains('a'); // kiểm tra xem Set có chứa phần tử a
không
```

Một số cấu trúc dữ liệu phức tạp

- Dart xây dựng thêm các loại cấu trúc phức chứa trong thư viện dart: collection
- Trước khi sử dụng cần nạp thư viện `import 'dart:collection';`

Queue	Hàng đợi Queue chỉ tương tác với cấu trúc này ở đầu và cuối hàng đợi
HashMap	Sử dụng và ý nghĩa giống Map, có điều các đối tượng làm key phải cung cấp sự so sánh == thông qua giá trị .hashCode
HashSet	Sử dụng và ý nghĩa giống Set, so sánh == thông qua giá trị .hashCode
LinkedList	Sử dụng và ý nghĩa giống List, chỉ có điều nó lưu trữ dữ liệu theo cách tối ưu để duyệt, tìm kiếm nhanh, tốt để quản lý dữ liệu như chèn và xoá một lượng lớn phần tử

Một số tài liệu tham khảo

Dart SDK for MacOS: http://dartdoc.takyam.com/downloads/mac.html
Trang chủ của Dart: https://dart.dev/
Thư viện packages: https://pub.dev/