**BÁO CÁO TUẦN 2**

**Họ và tên:** Nguyễn Kỳ Anh  
**MSSV:** 20225793  
 **GR1**  
**Giáo viên hướng dẫn:** Nguyễn Đức Toàn

1. **Hàng đợi – Queue**

- Với hàng đợi Queue thì chỉ tương tác với cấu trúc này ở đầu hàng đợi và cuối hàng đợi (thêm / bớt).

|  |  |
| --- | --- |
| add, addLast | Thêm phần tử vào cuối hàng add(ele); hoặc addLast(ele) |
| addFirst | Thêm phần tử vào đầu hàng addFirst(ele); |
| removeFirst() | Xóa phần tử đầu tiên |
| removeLast() | Xóa phần tử cuối cùng |
| remove() | Xóa 1 phần tử remove(ele) |

**VD:**

var q = new Queue();

q.add('A');

q.add('B');

q.addFirst('A0');

q.addLast('B0');

print(q);

q.removeFirst();

q.removeLast();

q.remove('B');

**2. Convert giữa các cấu trúc dữ liệu: List, Maps, Sets, Queue trong Dart**

- List:

* Là một danh sách có thứ tự (ordered collection), tương tự như mảng trong các ngôn ngữ khác.
* Có thể chứa các phần tử trùng lặp.
* Ví dụ: List<int> list = [1, 2, 2, 3];

- Map:

* Là một tập hợp các cặp key-value (khóa-giá trị), trong đó key không trùng lặp.
* Dùng để lưu trữ dữ liệu dưới dạng ánh xạ.
* Ví dụ: Map<int, String> map = {1: 'a', 2: 'b'};

- Set:

* Là một tập hợp không có thứ tự (unordered collection), không chứa các phần tử trùng lặp.
* Dùng để lưu trữ các giá trị duy nhất.
* Ví dụ: Set<String> set = {'a', 'b', 'c'};

- Queue:

* Là một hàng đợi, hỗ trợ thêm/xóa phần tử ở cả đầu và cuối (FIFO hoặc LIFO).
* Thường được sử dụng trong các bài toán xử lý tuần tự.
* Ví dụ: Queue<int> queue = Queue.from([1, 2, 3]);

**2.1. Từ List sang các cấu trúc khác**

* List → Set: toSet()
* List → Map: asMap(), Map.fromEntries() (kết hợp với map() để ánh xạ tùy chỉnh)
* List → Queue: Queue.from()

**2.2. Từ Set sang các cấu trúc khác**

* Set → List: toList()
* Set → Map: Map.fromEntries() (kết hợp với map() để ánh xạ tùy chỉnh)
* Set → Queue: Queue.from()

**2.3. Từ Map sang các cấu trúc khác**

* Map → List: keys.toList(), values.toList(), entries.toList()
* Map → Set: keys.toSet(), values.toSet()
* Map → Queue: Queue.from(map.keys), Queue.from(map.values)

**2.4. Từ Queue sang các cấu trúc khác**

* Queue → List: toList()
* Queue → Set: toSet()
* Queue → Map: Map.fromEntries() (kết hợp với toList().asMap().entries.map() để ánh xạ tùy chỉnh)

3.

Trong các ngôn ngữ lập trình, vòng lặp cho phép một khối mã được thực thi lặp đi lặp lại nhiều lần.

Ngôn ngữ **Dart** cung cấp các loại vòng lặp sau:

* for
* for..in
* while
* do..while

Các câu lệnh (statement) **break** và **continue** có thể xuất hiện bên trong vòng lặp:

**break**

Khi bắt gặp lệnh **break** chương trình sẽ kết thúc vòng lặp ngay lập tức.

**continue**

Khi bắt gặp lệnh **continue** chương trình sẽ bỏ qua các dòng lệnh phía dưới của ***continue***và thực thi bước lặp tiếp theo (Nếu điều kiện của vòng lặp vẫn đúng).

* + Vd vòng lặp for:

**void** main() {

print('For loop example');

**for** (**var** i = 0; i < 10; i = i + 3) {

print('i= $i');

}

}

* + VD vòng lặp for in:

**void** main() {

// Create a List:

**var** names = ['Tom', 'Jerry', 'Donald'];

**for** (**var** name **in** names) {

print('name: $name');

}

}

* + VD vòng lặp while:

**void** main() {

print('While loop example');

**var** x = 2;

**while** (x < 10) {

print('Value of x = $x');

x = x + 3;

}

print('Done!');

}

* + VD vòng lặp do while:

**void** main() {

**var** value = 3;

**do** {

print('Value = $value');

value = value + 3;

} **while** (value < 10);

}

**4. Function**

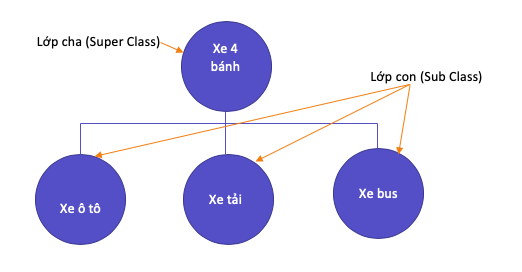
Các cách khai báo hàm

* Hàm có tên: Khai báo với tên rõ ràng, có thể trả về giá trị (ví dụ: int add(int a, int b) { return a + b; }) hoặc không (void sayHello(String name) { print('Hello, $name!'); }).
* Hàm mũi tên: Ngắn gọn cho biểu thức đơn (int add(int a, int b) => a + b;).
* Hàm ẩn danh: Không tên, dùng trong forEach hoặc map (ví dụ: list.forEach((item) => print(item \* 2));).
* Hàm với tham số tùy chọn: Sử dụng [] (positional) hoặc {} (named) (ví dụ: String greet(String name, [String greeting = 'Hello']) => '$greeting, $name!';).
* Hàm gán cho biến: Gán hàm cho biến để sử dụng như đối tượng (var add = (int a, int b) => a + b;).
* Hàm với kiểu dynamic: Trả về nhiều kiểu dữ liệu (dynamic getValue(bool isNumber) => isNumber ? 42 : 'Hello';).

**5. Tính kế thừa**

Trong một phần mềm hay chương trình, được cấu tạo bởi nhiều lớp khác nhau cùng các thành phần khác.

Mối quan hệ giữa các lớp, có mối quan hệ kế thừa, gồm lớp cha ( super class )và các lớp con(sub class), Các lớp con đó lại có thể là lớp cha của các lớp khác.



Mục đích của kế thừa là tái sử dụng.  
Lớp con có thể sở hữu các thuộc tính và phương thức public của lớp cha nhưng không được sở hữu các thuộc tính hay phương thức private và các hàm constructor.

Biểu diễn kế thừa trong Dart cũng tương tự như trong Java : dùng extends

ví dụ:

|  |  |
| --- | --- |
| 1  2 | class Car extends XeBonBanh{...}  class Truck extends XeBonBanh{...} |

**6. Abstract (Lớp trừu tượng)**

Lớp trừu tượng trong ngôn ngữ Dart là lớp có hành vi chưa được xác định rõ. Cách nhận biết là có từ khoá abstract ở trước class :  
Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5 | abstract class Hinh{    double canh;    double tinhChuVi();    double tinhDienTich();  } |

– Ghi đè phương thức : Overriding – là trườnh hợp khi lớp con muốn sửa phương thức của lớp cha. Đặc biệt nếu lớp cha có những phương thức trừu tượng thì việc ghi đè là bắt buộc.

– Từ khoá super trong kế thừa: Dùng để truy cập đến các biến, hàm ( public) và cả constructor của lớp cha.

VD:

|  |
| --- |
| // Lớp cha    class LopCha {      void sendMessage(){    print ("Message từ lớp cha");    }  // Lớp con  import 'package:dart\_language/lop\_cha.dart';  class LopCon extends LopCha{    @override    void sendMessage() {      // TODO: implement sendMessage      super.sendMessage();      print ("Message từ lớp con");    }  }  // Sử dụng trong main.dart    import 'package:dart\_language/lop\_con.dart';  void main() {    var lopCon = LopCon();    lopCon.sendMessage();  } |

Kết quả:

****

**7. Tính đa hình**

**\* Tính đa hình**  
Tính đa hình trong ngôn ngữ Dart cũng có ý nghĩa giống như trong các ngôn ngữ khác. Cùng biểu diễn một hành vi nhưng từng lớp có cách biểu diễn khác nhau.

Đoạn code sau là một ví dụ:

|  |  |
| --- | --- |
|  | // Khai báo về tính năng của Xe  abstract class Xe {    void chuyenCho(); // Chuyên chở  }  // Khai báo của Xe tải  import 'package:dart\_language/XeCo.dart';  class XeTai extends Xe{    @override    void chuyenCho() {      print (" Chỉ chở hàng");    }  }  // Khai báo về Xe khách  import 'package:dart\_language/XeCo.dart';  class XeKhach extends Xe{    @override    void chuyenCho() {      print (" Chỉ chở người");    }  } |

Như vậy, Cùng một hành vi là Chở, nhưng với các loại xe là các công dụng lại khác nhau – Đó là tính đa hình.

**8. Mixin programming**

- Mixin là một cách định nghĩa mã có thể được sử dụng lại trong nhiều hệ thống phân cấp lớp. Chúng được thiết kế để cung cấp các triển khai thành viên hàng loạt.

- Để sử dụng mixin, hãy sử dụng with từ khóa theo sau là một hoặc nhiều tên mixin. Ví dụ sau đây cho thấy hai lớp sử dụng (hoặc, là các lớp con của) mixin:

class Musician extends Performer with Musical {

// ···

}

class Maestro extends Person with Musical, Aggressive, Demented {

Maestro(String maestroName) {

name = maestroName;

canConduct = true;

}

}

- Để định nghĩa một mixin, hãy sử dụng mixinkhai báo. Trong trường hợp hiếm hoi khi bạn cần định nghĩa cả mixin *và* class, bạn có thể sử dụng [mixin class khai báo](https://dart.dev/language/mixins" \l "class-mixin-or-mixin-class) .

Mixin và các lớp mixin không thể có extends mệnh đề và không được khai báo bất kỳ hàm tạo nào.

Ví dụ:

mixin Musical {

bool canPlayPiano = false;

bool canCompose = false;

bool canConduct = false;

void entertainMe() {

if (canPlayPiano) {

print('Playing piano');

} else if (canConduct) {

print('Waving hands');

} else {

print('Humming to self');

}

}

}

* + Chỉ định các thành viên mà mixin có thể gọi đến chính nó

Đôi khi một mixin phụ thuộc vào khả năng gọi một phương thức hoặc truy cập các trường, nhưng không thể tự xác định các thành viên đó (vì mixin không thể sử dụng tham số xây dựng để khởi tạo các trường của riêng chúng).

- Các phần sau đây đề cập đến các chiến lược khác nhau để đảm bảo bất kỳ lớp con nào của mixin đều xác định bất kỳ thành viên nào mà hành vi của mixin phụ thuộc vào.

Xác định các thành viên trừu tượng trong mixin

- Việc khai báo một phương thức trừu tượng trong mixin sẽ buộc bất kỳ kiểu nào sử dụng mixin phải xác định phương thức trừu tượng mà hành vi của nó phụ thuộc vào.

VD:

mixin Musician {

void playInstrument(String instrumentName); // Abstract method

void playPiano() {

playInstrument('Piano');

}

void playFlute() {

playInstrument('Flute');

}

}

class Virtuoso with Musician {

@override

void playInstrument(String instrumentName) { // Subclass must define.

print('Plays the $instrumentName beautifully');

}

}

* + Triển khai một giao diện

Tương tự như việc khai báo tóm tắt mixin, việc đặt một implementsmệnh đề vào mixin trong khi không thực sự triển khai giao diện cũng sẽ đảm bảo mọi phụ thuộc thành viên đều được xác định cho mixin.

VD:

abstract interface class Tuner {

void tuneInstrument();

}

mixin Guitarist implements Tuner {

void playSong() {

tuneInstrument();

print('Strums guitar majestically.');

}

}

class PunkRocker with Guitarist {

@override

void tuneInstrument() {

print("Don't bother, being out of tune is punk rock.");

}

}

* + Sử dụng onmệnh đề để khai báo một siêu lớp

Mệnh đề này ontồn tại để xác định loại mà supercác cuộc gọi được giải quyết. Vì vậy, bạn chỉ nên sử dụng nó nếu bạn cần có một supercuộc gọi bên trong một mixin.

Mệnh đề này onbuộc bất kỳ lớp nào sử dụng mixin cũng phải là lớp con của loại trong onmệnh đề. Nếu mixin phụ thuộc vào các thành viên trong siêu lớp, điều này đảm bảo các thành viên đó có sẵn khi mixin được sử dụng.

VD:

class Musician {

musicianMethod() {

print('Playing music!');

}

}

mixin MusicalPerformer on Musician {

performerMethod() {

print('Performing music!');

super.musicianMethod();

}

}

class SingerDancer extends Musician with MusicalPerformer { }

main() {

SingerDancer().performerMethod();

}