Dart

Class

Class

- Every object is an instance of a class, and all classes descend from Object

- Mixin-based inheritance - every class (except for Object) has exactly one superclass

Instance variables

- Declare
- getter and setter

```dart
class Point {
  num x;
  num y;
}

void main() {
  var point = Point();
  point.x = 4; // Use the setter method for x.
  assert(point.x == 4); // Use the getter method for x.
  assert(point.y == null); // Values default to null.
}
```

Class variables and methods

- Use the static keyword to implement class-wide variables and methods

- Static variables aren’t initialized until they’re used

```
class Queue {
  static const initialCapacity = 16;
  // ···
}

void main() {
  assert(Queue.initialCapacity == 16);
}
```

Static methods

```dart
class Point {
  num x, y;
  Point(this.x, this.y);

  static num distanceBetween(Point a, Point b) {
    var dx = a.x - b.x;
    var dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
  }
}

void main() {
  var a = Point(2, 2);
  var b = Point(4, 4);
  var distance = Point.distanceBetween(a, b);
```

Constructors

- var p1 = Point(2, 2);
  var p1 = Point.fromJson({ ‘x’ : 1 , ‘y’: 2 });
- var p = const ImmutablePoint(2, 2);
- print('The type of p1 is ${p1.runtimeType}’);

Constructors

- Default constructors
- Named constructors
- Constructors aren’t inherited
- Initializer list

Constructors

- Redirecting constructors
- Constant constructors: define a const constructor and make sure that all instance variables are final

- Factory constructors (https://stackoverflow.com/questions/53886304/understanding-factory-constructor-code-example-dart)

Method
Instance methods

```dart
import 'dart:math';

class Point {
  num x, y;

  Point(this.x, this.y);

  num distanceTo(Point other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return sqrt(dx * dx + dy * dy);
  }
}
```

Instance methods

Method
Getters and setters

```dart
class Rectangle {
  num left, top, width, height;

  Rectangle(this.left, this.top, this.width, this.height);

  // Define two calculated properties: right and bottom.
  num get right => left + width;
  set right(num value) => left = value - width;
  num get bottom => top + height;
  set bottom(num value) => top = value - height;
}

void main() {
  var rect = Rectangle(3, 4, 20, 15);
  assert(rect.left == 3);
  rect.right = 12;
  assert(rect.left == -8);
}
```

getters、setters

Method
Abstract methods

```
abstract class Doer {
  // Define instance variables and methods...

  void doSomething(); // Define an abstract method.
}

class EffectiveDoer extends Doer {
  void doSomething() {
    // Provide an implementation, so the method is not abstract here...
  }
}
```

Abstract classes

- a class that can’t be instantiated

- useful for defining interfaces, often with some implementation

Implicit interfaces

- If you want to create a class A that supports class B's API without inheriting B's implementation, class A should implement the B interface

- class A implements B

Extending a class

```
class Television {
  void turnOn() {
    _illuminateDisplay();
    _activateIrSensor();
  }
  // ...
}

class SmartTelevision extends Television {
  void turnOn() {
    super.turnOn();
    _bootNetworkInterface();
    _initializeMemory();
    _upgradeApps();
  }
  // ...
}
```

Overriding members

```
class SmartTelevision extends Television {
  @override
  void turnOn() {...}
  // ···
}
```

Override operator

```dart
class Vector {
  final int x, y;

  Vector(this.x, this.y);

  Vector operator +(Vector v) => Vector(x + v.x, y + v.y);
  Vector operator -(Vector v) => Vector(x - v.x, y - v.y);

  // Operator == and hashCode not shown. For details, see note below.
  // ...
}

void main() {
  final v = Vector(2, 3);
  final w = Vector(2, 2);

  assert(v + w == Vector(4, 5));
  assert(v - w == Vector(0, 1));
}
```

Override operator

```dart
  var t2 = new Todo('dart', 10);

  print(t1.hashCode);
  print(t2.hashCode);

  print(t1 == t2); //Test for Equality of two objects
  print(identical(t1, t2)); //Test for identicality of two objects
}

class Todo {
  String todo;
  int priority;
  Todo(this.todo, this.priority);

  //Overriden the == operator
  bool operator ==(o) => o is Todo && o.todo == todo && o.priority ==
priority;
  //Overriden the get hashCode method
  int get hashCode => todo.hashCode^priority.hashCode;
}
```

```
529459523
529459523
true
false
```

https://medium.com/@ayushpguptaapg/demystifying-and-hashcode-in-dart-2f328d1ab1bc