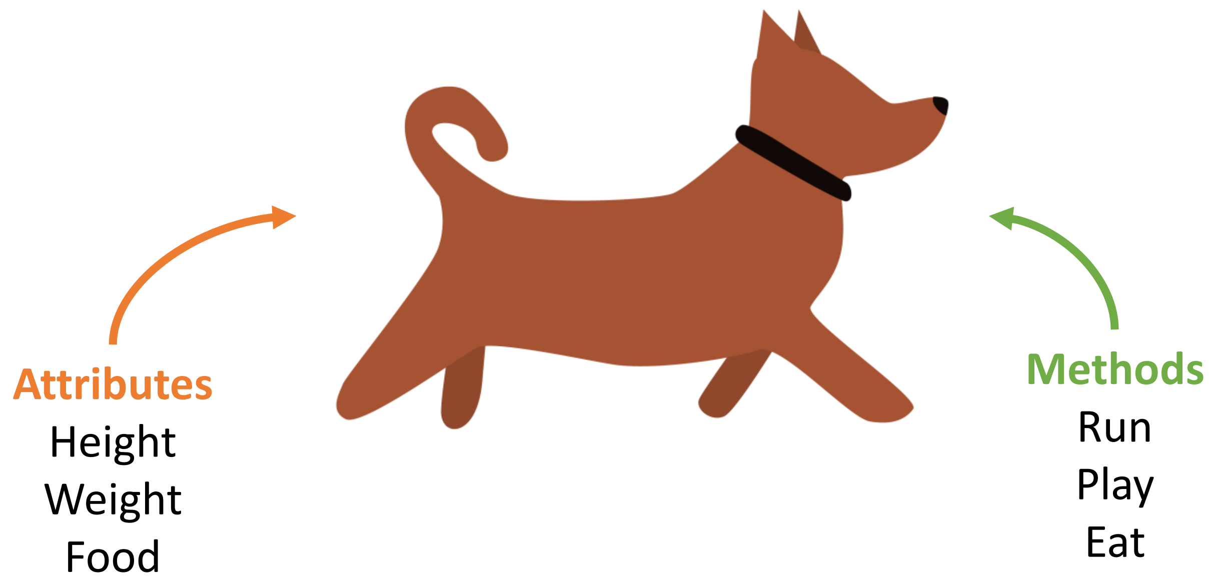
# MOBILE DEVELOPMENT

## W2-S1 – Dart – Object Oriented



CADT IDT

## Course Objectives

- ✓ Create a **class** with attributes method, constructors
- ✓ Build **immutable** objects
- ✓ Build objects with **optional arguments**
- ✓ Create objects using **different named constructors**
- ✓ Overload the **operator +**
- ✓ Provide **dynamically computed** attributes
- ✓ Build **aggregation of objects**

APPRENTICE

OF OBJECTS BUILDING

**LEVEL 1**

## Create a class **Point**

- ✓ A point has a **X and Y** position
- ✓ The class should provide a **method to translate** the point (dx, dy)

- ✓ *Test your class in different situations !*

*OBJECTIVES*

- ✓ Create a **class** with attributes method, constructors

*SOME HELP !*

https://dart.dev/language/classes

## EXPLANATIONS

APPRENTICE

OF OBJECTS BUILDING

LEVEL 1

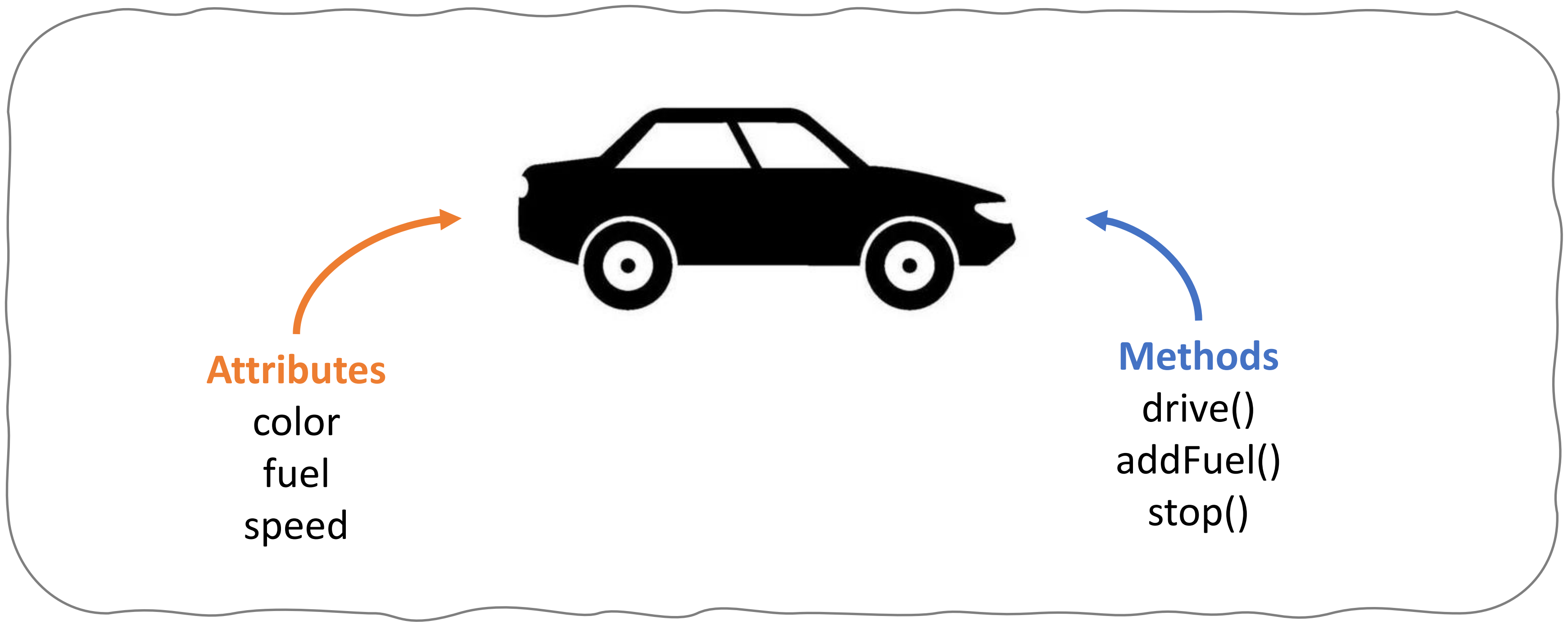
```
class Point {
  int x;
  int y;

  Point(this.x, this.y);

  @override
  String toString() {
    return "x= $x - y= $y";
  }
   void translate(int dx, int dy) {
    x+=dx;
    y+-dy;
   }
}

Run | Debug
main() {
  Point p1 = Point(1, 2);
  print(p1);
  p1.translate(10, 10);
  print(p1);
}
```

# Dart is an **object-oriented language**



*Thanks to classes, I can gather my car attributes and functions  in a same place*

EXPLAIN

# Dart classes : **key concepts**

All classes *except Null* descend from **Object**.

class

Class Attribute
Not final, not constant, not private...
*Everyone can change it for now!*

Constructor

Argument Y

Instance attribute Y

Class Method

```
class Point {
  int x;
  int y;

  Point(x, y) {
    this.x = x;
    this.y = y;
  }

  void translate(int dx, int dy) {
    x+=dx;
    y+-dy;
   }
}
```

```
main() {
   Point p1 = Point(1,2);
   Point p2 = new Point(3,4);
   p1.translate(10,20);
   p2.x = 50;
}
```

**instantiate a class**

New key word is optional

We can change x attribute freely

We invoke translate method on p1

EXPLAIN

# 3 way to manage your **constructors**

```
Point(x, y) {
              this.x = x;
              this.y = y;
            }
```

Constructor body

Initializer list

```
Point(int x, int y): this.x=x,  this.y=y;
```

```
Point(this.x, this.y);
```

Short form constructor !

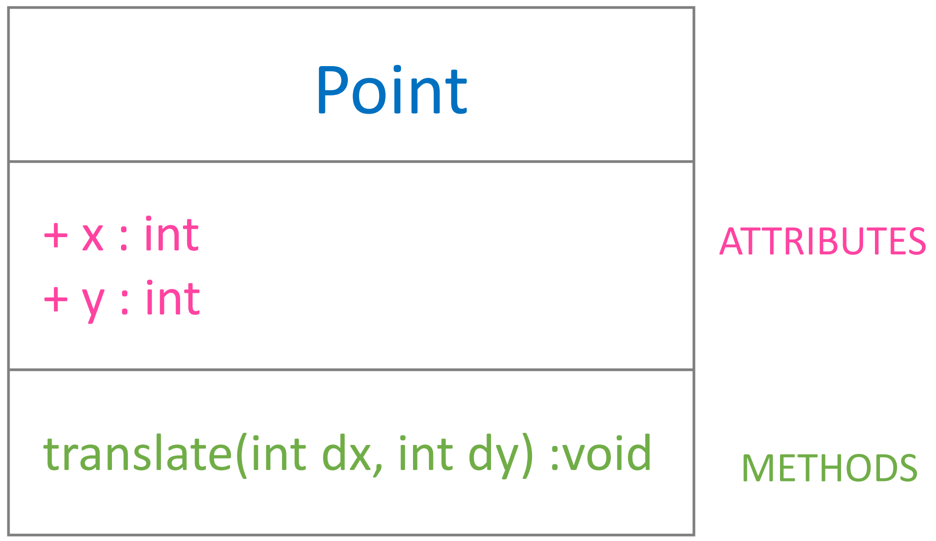
EXPLAIN

## We use **UML** to express the class definition

```
class Point {
  int x;
  int y;

  Point(x, y) {
    this.x = x;
    this.y = y;
  }

  void translate(int dx, int dy) {
    x+=dx;
    y+-dy;
   }
}
```



*We omit constructors when they are obvious*
*For clarify purpose*

NINJAS
OF THE IMMUTABILITY

LEVEL 2

## Make your Point **immutable !!**

- ✓ We should **NEVER be able to change** the class attributes !

*Example: maybe your translate method should, return a new object instead of changing the current object…*

- ✓ *Test your class in different situations !*

*OBJECTIVES*

- ✓ Make a class **immutable**

*SOME HELP !*

- ✓ Understand immutability

# EXPLANATIONS

NINJAS

OF THE IMMUTABILITY

LEVEL 2

```
class Point {
  final int _x;
  final int _y;

  Point(this._x, this._y);

  @override
  String toString() {
    return "x= $_x - y= $_y";
  }

  Point translate(int dx, int dy) {
    return Point(_x + dx, _y + dy);
  }

   get x => _x;
   get y => _y;
}

Run | Debug
main() {
  Point p1 = Point(1, 2);
  print(p1);
  Point p2 = p1.translate(10, 10);
  print(p1);
  print(p2);
}
```

EXPLAIN

## What are the **4 changes** to make a class **immutable**?

MUTABLE

```
class Point {
  int x;
  int y;

  Point(this.x, this.y);

  void translate(int dx, int dy) {
    x+=dx;
    y+-dy;
   }
}
```

IMMUTABLE

```
class Point {
  final int _x;
  final int _y;

  Point(this._x, this._y);


   Point translate(int dx, int dy) {
    return Point(_x+dx, _y+dy);
   }

   get x => _x;
   get y => _y;
}
```

*Note : do we really need private access to x and y here to maintain immutability? Why?*

ARCHITECT

OF NAMED ARGUMENTS

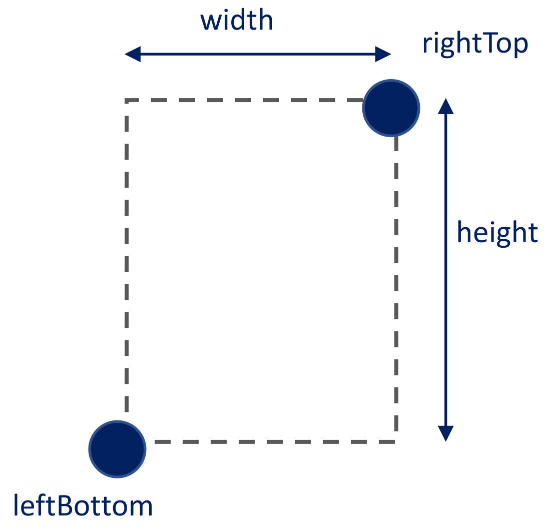
LEVEL 3

## Create a class **Shape**

- ✓ A shape has an **left-bottom point** (of type Point)
- ✓ A shape has also a **width and height**
- ✓ The shape can have  a **background color**, but it s **optional**

- ✓ We also want to know the **right-top point** (type Point)

- ✓ *Test your class in different situations !*



### *OBJECTIVES*

- ✓ Use named parameters
- ✓ Use nullable types
- ✓ Use getters for dynamic properties

### *SOME HELP  !*

- ✓ Understand named parameters
- ✓ Understand the nullable ? Syntax
- ✓ Understand getter in Dart

# EXPLANATIONS

ARCHITECT

OF NAMED ARGUMENTS

LEVEL 3

```dart
enum Color { blue, red, green, yellow }

class Shape {
  final Point leftBottom;
  final int width;
  final int height;
  Color? backgroundColor;

  Shape(
      {required this.leftBottom,
      required this.width,
      required this.height,
      this.backgroundColor});

  Point get rightTop => Point(leftBottom.x + width, leftBottom.y + height);
}
```

We use a enum to represent the color

A class can be composed of other classes : this is **aggregation** (*or composition*)

We use **named argument** syntax To build our object

This is a getter in Dart

It provides a new (*dynamic*) attribute