Inelectronics
Students
Club

Flutter Workshop

Session 2

Presented by:

- Belkacem BEKKOUR
- Anes SEDDIK
- Abdelhamid RIAHI
- Anais ALEM
- Cheikh HASNI
- Hicham BOUHADI

Flutter

- DataTypes

- Flow statments

- Functions

- Object Orionted programming (OOP)

  - OOP Structure
  - Classes
  - Methods
  - Objects/ Constractor
  - Dart special Keywords

DataTypes

```
void main() {
int age = 16;
String name = "Inelectronics Students Club";
float number = 12.21;
Boolean state = True;
var num=[1,2,3,4];
var names ={Nadir , hamid, mohamed ,belkacem}
var president ={name:nadir,age:21}
}
```

Flow Statement

```dart
data_flow_2.dart > ...
Run | Debug
void main() {
    double avg = 15.5;
    if (avg >= 10) {
        print("You can pass");
    }

    int i = 0;
    while (i <= 10) {
        print(i);
        ++i;
    }

}
```

Functions

```dart
functions.dart > ...
Run | Debug
void main() {
  String printing(int age) {
    print("it is the ${age}th anniversary of ISC!");
    return "DONE";
  }


  printing(16);
}
```

Flow Statement

```dart
data_flow.dart > ...
Run | Debug
void main() {
  double avg = 15.5;
  if (avg >= 10) {
    print("You can pass");
  }

  for (int count = 0; count <= 10; count++) {
    print(count);
  }
}
```

Dart Object Oriented Programming

Object-oriented programming (OOP) is a programming method that uses objects and their interactions to design and program applications.

Advantages:

❖ It is easy to understand and use.

❖ It increases reusability and decreases complexity.

❖ It makes the code easier to maintain, modify and debug.

❖ It promotes teamwork and collaboration.

❖ It reduces the repetition of code.

Dart Object Oriented Programming

Features Of OOP :

❖ Class
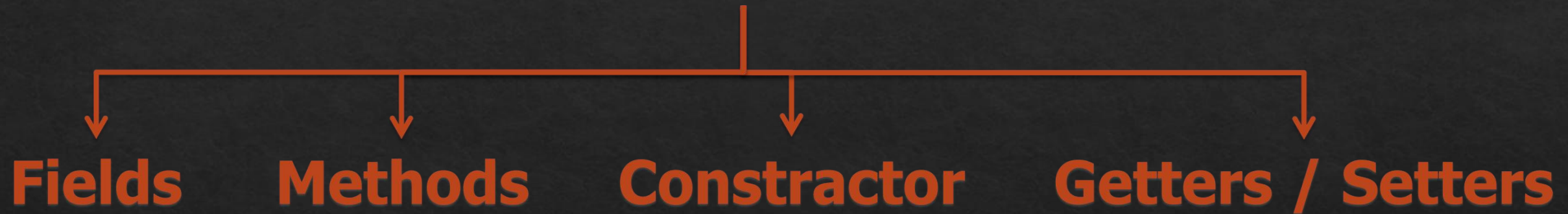❖ Object
❖ Encapsulation
❖ Inheritance
❖ Abstraction

Class In Dart

❖ A class defines the properties and methods that an object can have.

❖ **Fields:** class can contain many fields. Theses fields represnt any variable declared in the class.

❖ **Setters and Getters:** Allows the program to initialize and retrieve the values of the fields of a class. A default getter/ setter is associated with every class.

❖ **Constructors:** responsible for allocating memory for the objects of the class.

❖ **Methods:** Functions represent the actions/ behaviours that an object can take. They are also at times referred to as methods.

Class / Methods in dart

```dart
car.dart > ...
Run | Debug
void main(List<String> arguments) {
  var car1 = new Car();


  car1.move();
}


class Car {
  String color = "red";
  String name = "Fabia"; // Fields


  void move() {
    //Method
    print("the ${color} ${name} is moving");
  }
}
```

Obejct and Constactor

❖ **Object:** an object is an instance of the class.

☐ An object should have the same name as the class.
☐ We can create many objects from the same class.
☐ Syntax: Object_name = new Constractor_name();

❖ **Constractor:** The constactor is responsible about allocating memory for the objects of the class.

☐ The constractor should have the same name as the class.
☐ There are 3 Types of constractor: **Default - Parameterized- Named.**

Default Constactor


```dart
car_constractor.dart > main
Run | Debug
void main(List<String> arguments) {
    var car1 = new Car();

    print("This is the code of the main function");

    car1.move();
}
```

```dart
class Car {
    String color = "red";
    String name = "Fabia"; // Fields

    //Default Constractor
    Car() {
        print("This is the default constractor");
    }


    void move() {
        //Method
        print("The ${color} ${name} is moving");
    }
}
```

Parameterized Constactor


```dart
car_constractor.dart > Car
Run | Debug
void main(List<String> arguments) {
  var car1 = new Car("Blue", "2008");

  print("This is the code of the main function");

  car1.move();
}
```

```dart
class Car {
  String color = "red";
  String name = "Fabia"; // Fields

  //Default Constractor
  // Car() {
  //    print("This is the default constractor");
  // }

  //Parametrized Constractor
  Car(this.color, this.name) {}

  void move() {
    //Method
    print("The ${color} ${name} is moving");
  }
}
```

Inelectronics Students Club

Named Constactor


```dart
car_constractor.dart > Car
Run | Debug
void main(List<String> arguments) {
  var car1 = new Car.color_only("Blue");

  print("This is the code of the main function");

  car1.move();
}
```

```dart
class Car {
  String color = "red";
  String name = "Fabia"; // Fields

  //Default Constractor
  // Car() {
  //   print("This is the default constractor");
  // }

  //Named Constractor
  Car.color_only(this.color) {}

  void move() {
    //Method
    print("The ${color} ${name} is moving");
  }
}
```

Inelectronics Students Club

Getters / Setters

```
getter_setters.dart > ⬡ main
Run | Debug
void main(List<String> arguments) {
  var car1 = new Car("Blue", "208");
  var car2 = new Car("Black", "202");

  car1.move();
  car2.move();


  car1.color = "Black";


  car1.move();
}
```

```dart
class Car {
  String color = "red";
  String name = "Fabia"; // Fields

  //parametrized constractor
  Car(this.color, this.name) {}

  String get get_color {
    return color;
  }

  void set age(int name) {
    name = name;
  }

  void move() {
    //Method
    print("the ${color} ${name} is moving");
  }
}
```

Dart Inheritance

Dart inheritance is defined as the process of deriving the properties and characteristics of another class.

```dart
car_constractor.dart > ...
Run | Debug
void main(List<String> arguments) {
  //var car1 = new Car.color_only("Blue");
  var Bwm1 = new BMW(true, "Optical type");
  print("This is the code of the main function");

  //car1.move();
}
```

```dart
//Extending a class
class BMW extends Car {
  bool electrical_model = true;
  String fingure_print;

  BMW(this.electrical_model, this.fingure_print) {}

  String print_information(bool electrical_model, String fingure_print) {
    print("model information: ${electrical_model}, ${fingure_print}");
    return "Infromation";
  }

}
```

The mother class is called the abstract class.

Dart Special Keywords

◈ final:

  ◈ Final Method: We can't override on it.

  ◈ Final variable: Constant variable the only difference between **Const** and final will be in compilation part.

◈ Static:

  ◈ Static class: Can't be instantiated.

  ◈ Static variable: Allocated only one in the memory

  ◈ Static Method: To access the static method, we don't need to create a class instance. Can only use static variables.