University of Padova
Department of Information Engineering

Biomedical Wearable Technologies
for Healthcare and Wellbeing

Dart 101 – Part 2

A.Y. 2021-2022

Giacomo Cappon

Outline

- **Recap**
- Classes
- Inheritance
- Other things
- Asynchrony

- Exercises
- Homework
- Resources

Recap

Lesson 1
Collaborate and version code ✓

Lesson 2/3
Get familiar with Dart

Lesson 4
Understand Flutter's principles

Lesson 5
Create a layout

Lesson 6
Navigate between screens

Lesson 7
Manage the app state

Milestone #1

Lesson 8
Make simple API calls

Lesson 9
Fetch wearable data

Milestone #2

Lesson 10
Implement user authentication

Lesson 11
Persist user data

Lesson 12
Advanced stuff

Milestone #3

Do something with your fantasy

Recap

- What is Dart?
  - Dart is a object-oriented, open source, and reactive language
  - It is pretty new (2011)
  - Cross-platform oriented

- What we learned last time?
  - How to write and run programs in Dart
  - Dart’s synthax
  - How to write functions in Dart

- Today we will dive into aspects related to object-oriented programming (OOP).

Outline

- Recap
- **Classes**
- Inheritance
- Other things
- Asynchrony

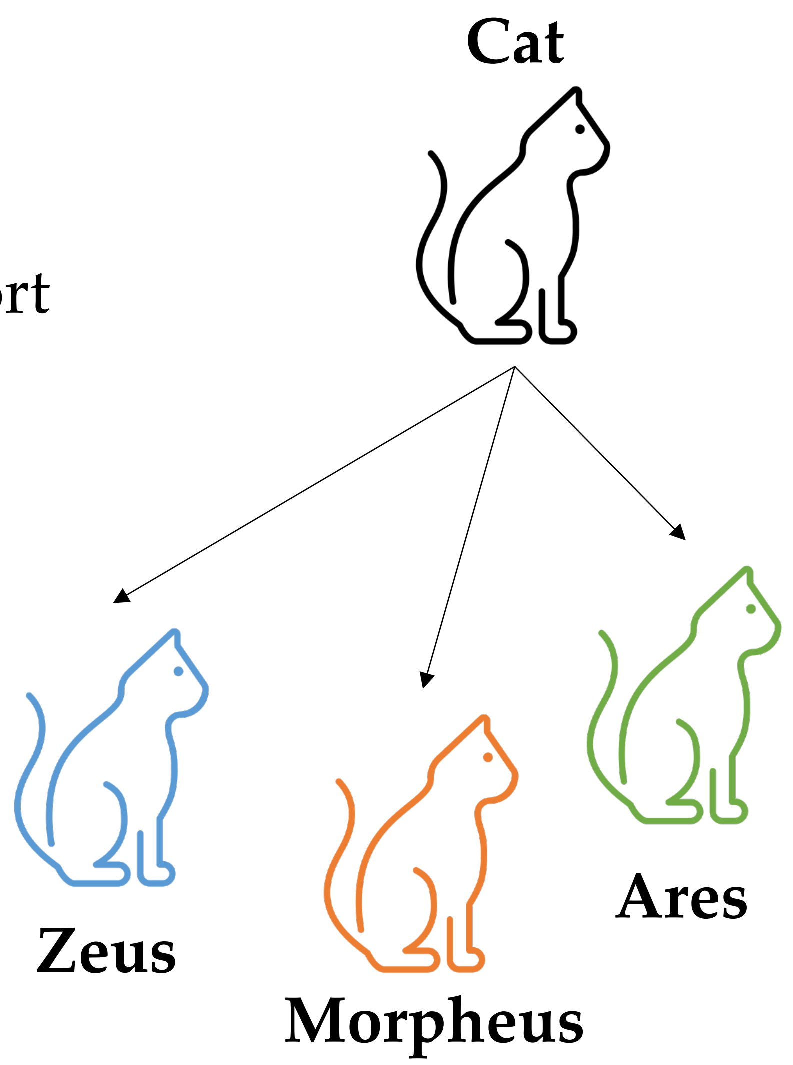
- Exercises
- Homework
- Resources

Recap: Classes and Objects

- The core concepts in every object-oriented programming (OOP) language: **classes** and **objects**.

- A class is a sort of blueprint for creating objects (a sort of data structure), providing initial values for state (defined by a set of instance variables), and implementations of behavior (defined by a set of methods)

- An object is an instance of a class.



How to define a class

- A class can be defined using the synthax:

```
class className{
        listOfInstanceVariables;
        listOfConstructors;
        listOfMethods;
}
```

Used to define the state of an object

Used to create objects

Used to define the behaviour of an object (what we can do with it)

How to define a class

- Let's try to create a class for Animals

```
class Animal{
     double? weight;

     String? name;

}//Animal
```

Note: Instance variables that are uninitialized have the value null (that's why we put the **?** there)

Create (construct) an object

- To create objects of a class we have to define constructors: special methods that are used for the purpose. In Dart, constructors are:
  - **Unnamed**: using the synthax
    `ClassName(parameterList){}`
  - **Named**: using the synthax
    `ClassName.name(parameterList){}`

- Each class can have **1 unnamed** constructor, and **multiple named** constructors.

Constructors

- Let’s try to create a class for Animals

```
class Animal{
     double? weight;
     String? name;

     //Unnamed (default) constructor
     Animal();

     //Named constructor 1
     Animal.withName(this.name);

     //Named constructor 2
     Animal.withWeight({this.weight});

     //Named constructor 3.
     Animal.fuffy() : name = 'Fuffy', weight = 2;

}//Animal
```

Note: This is equivalent to write

```
Animal(String? name){
   this.name = name;
}//Animal
```

Constructors follows the same synthax rules as functions regarding parameters.

It is possible to use the so-called “Initializer list”

Create and use objects

- Then we can finally create and use objects!

```
void main(List<String> args) {

    var animal = Animal();

    var animal2 = Animal.withName('GoodBoy');
    var animal3 = Animal.withWeight(weight: 10);
    var animal4 = Animal.fuffy();

    animal.weight = 100;
    print(animal.weight); // This will print '100.0'
    assert(animal.name == null); // This will print 'null'

}//main
```

How to create an object using the unnamed constructor

How to create an object using the named constructors

Instance variables of animal can be accessed using the dot notation

Full example in lab_03-dart_101_part_2/01-class_definition_and_constructors.dart

Methods

- Methods defines the behaviour of an object. Defining a method is similar to defining a function:

```
class Car{

  //Instance variables can be final. In this case, they must be set only once (in the constructor).
  final String? manufacturer;
  bool? isEletric;
  int? mileageSinceRevision;

  //Constructors
  Car({this.manufacturer});
  Car.used({this.manufacturer, this.mileageSinceRevision});

  //A method that performs a revision of the Car
  void doRevision(){
    mileageSinceRevision = 0;
    //...other revision things...
  }//doRevision
}//Car
```

Using methods

➢ Similarly, using methods is pretty straightforward:

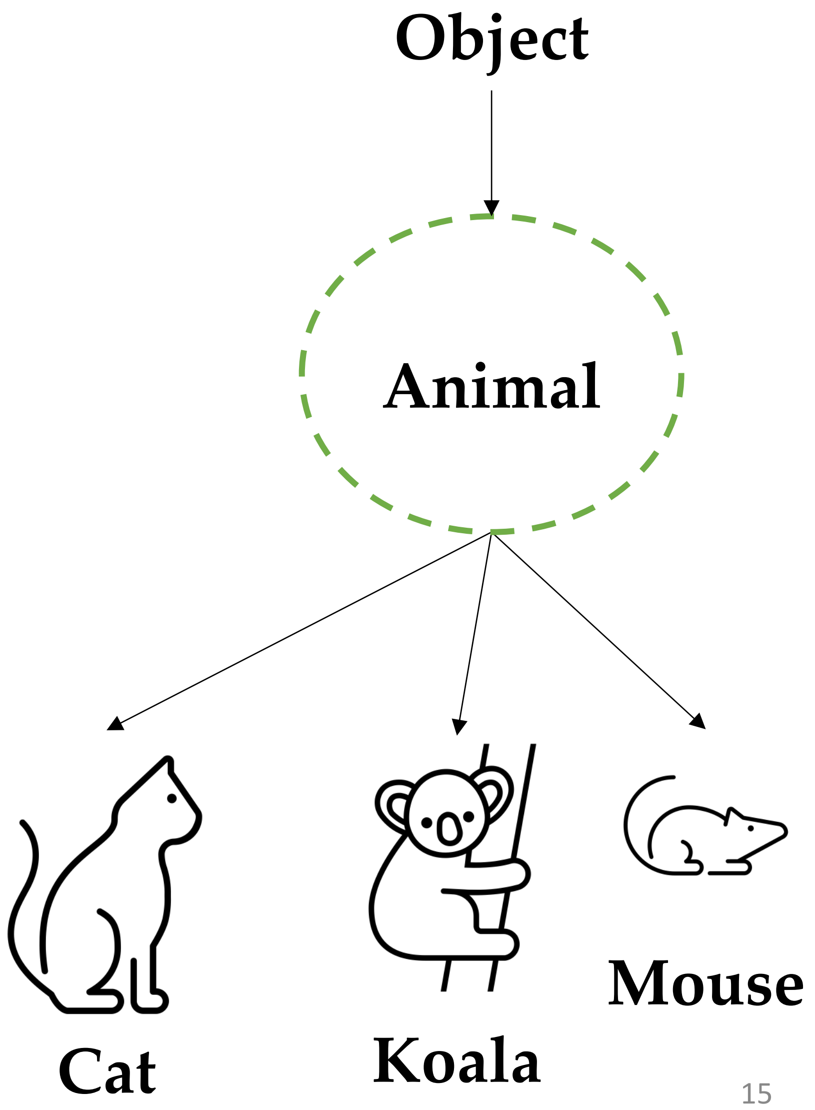
```
void main(List<String> args) {

  //Buy a used Ferrari that needs a revision
  var car = Car.used(manufacturer: 'Ferrari',
     mileageSinceRevision: 1000);

  //Do revision (methods can be used through the dot notation)
  car.doRevision();

  print(car.mileageSinceRevision); // This will print 0

}//main
```

Outline

Recap: Inheritance

- VERY important concepts in OOP: **inheritance**

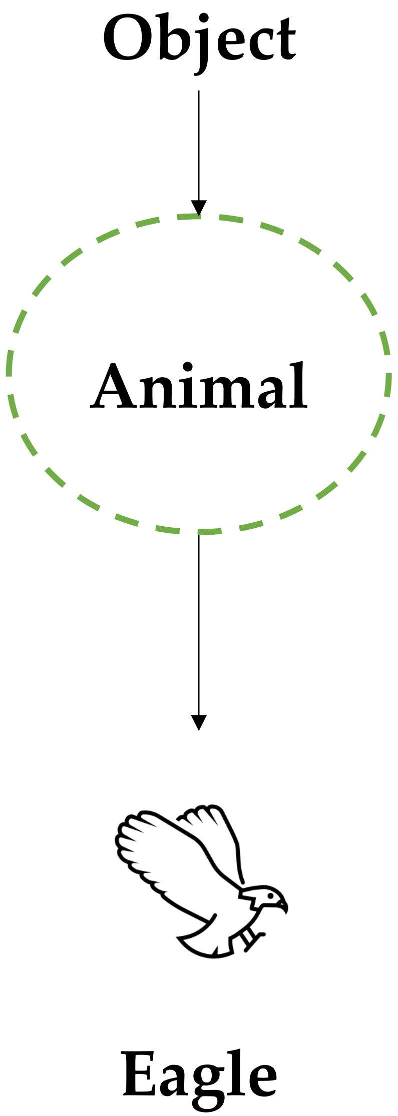
- A class can **extend** another (more generic) class the aim being:
  - Defining specific behaviors
  - Reusing the "superclass" code
  - Redefining (overriding) superclass' methods



Inheritance (example)

- Let's write the generic Animal class

```
class Animal{

  double? weight;
  String? name;

  void jump(){
    print('Jump');
  }//jump

  void eat(){
    print('Omnivorous');
  }//eat

}//Animal
```



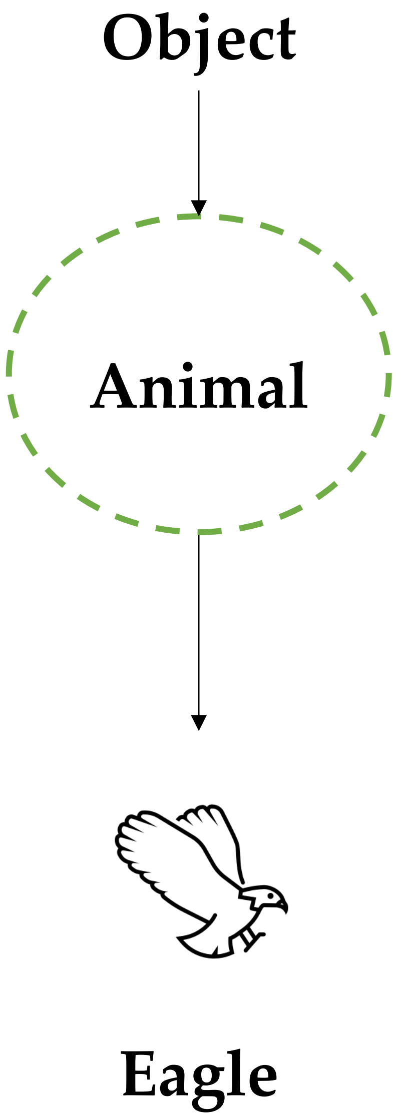
Inheritance (example)

- Let’s also redefine (override) a special method of the Object superclass: **toString()**

```
class Animal{

  double? weight;
  String? name;

  Animal();
  Animal.withName(this.name);

  void jump(){
    print('Jump’);
  }//jump

  void eat(){
    print('Omnivorous’);
  }//eat

  @override
  String toString() {
    return '(weight: $weight, name:
$name)’;
  }//toString

}//Animal
```

The override decorator is used to tell Dart that we are redefining a method of the superclass we are inheriting from

`toString()` is a special method that is called when we want to print the state of an object (see next slides…)



Inheritance (example)

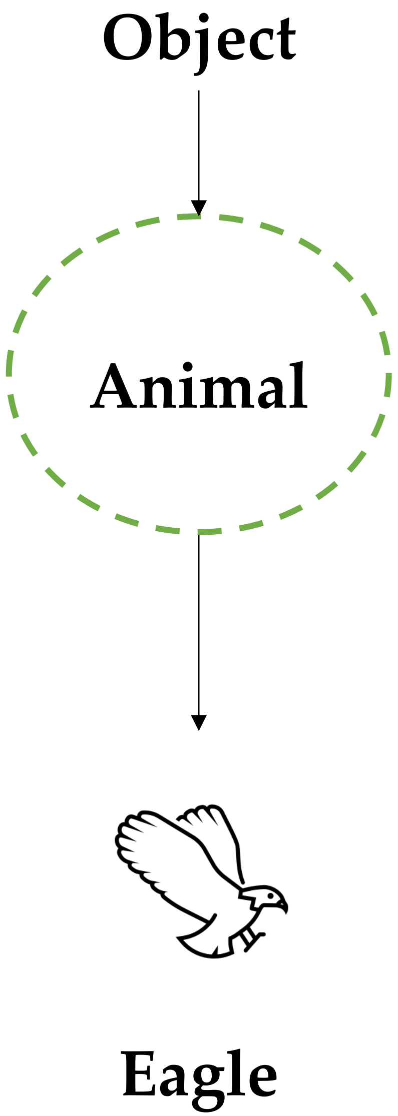
- Let's specify (extend) the Animal class:

```
class Eagle extends Animal{

  Eagle() : super();
  Eagle.withName(name) :
       super.withName(name);

void fly(){
    print('Fly');
  }//fly

  @override
  void eat(){
    print('Carnivorous');
  }//eat

  @override
  String toString() {
    return super.toString();
  }//toString
}//Eagle
```

Extend a class using the **extends** keyword

Note: in Animal we did not explicitly extend the **Object** class (this is automatic if not specified)

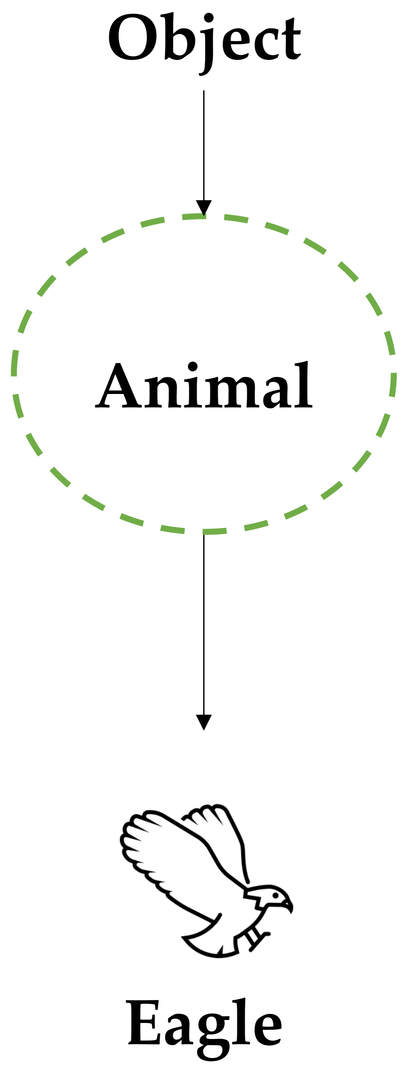
super is a special keyword that refers to the superclass



Inheritance (example)

- Let's use it

```
void main(List<String> args) {

  Animal animal = Animal();
  animal.jump();
  animal.eat();
  animal.weight = 10;
  print(animal);

  Eagle eagle = Eagle();
  eagle.jump();
  eagle.eat();
  eagle.name = 'Bob';
  print(eagle);

}//main
```



Full example in lab_03-dart_101_part_2/03-inheritance.dart

Outline

Other things…

- Unfortunately, we do not have time to review:
  - The 1000 ways to define and use constructors and methods
  - Enumerated types
  - Abstract classes
  - Interfaces
  - Generics
  - Mixins
  - Visibility
  - Libraries

- What's the spirit? If you'll need to use these concepts and you do not know how
  - It is very easy to find answers online (Google, Stackoverflow,…)
  - You can ALWAYS ask us

Outline

- Recap
- Classes
- Inheritance
- Other things
- **Asynchrony**

- Exercises
- Homework
- Resources

New concept: asynchrony

- Let's learn something (I believe) new.

- Dart (and Flutter) is full of asynchronous functions: they return after doing **something** possibly time consuming without waiting for that **something** to complete

- Common asynchronous operations:
  - Fetching data over the net
  - Writing/Reading data from a database
  - Load and show an image stored within the phone

- This is a problem because this

  -
    ```
    fetchDataFromFacebook(); // <-- asynchronous stuff
    print('Done');
    ```

  Can possibly print 'Done' before actually finishing fetching data!

- We need to learn how to manage asynchronous code in a synchronized fashion!

Key terms

- **synchronous operation**: A synchronous operation blocks other operations from executing until it completes.
- **synchronous function**: A synchronous function only performs synchronous operations.
- **asynchronous operation**: Once initiated, an asynchronous operation allows other operations to execute before it completes.
- **asynchronous function**: An asynchronous function performs at least one asynchronous operation and can also perform synchronous operations.

Future

- Dart manages asynchrony using the **`Future`** class

- A future (lower case "f") is an instance of the `Future` class. A future represents the result of an asynchronous operation, and can have two states: uncompleted or completed.

  - **Uncompleted**: When you call an asynchronous function, it returns an uncompleted future. That future is waiting for the function's asynchronous operation to finish or to throw an error.

  - **Completed**:
    - **With a value**: A future of type `Future<T>` completes with a value of type `T`. For example, a future with type `Future<String>` produces a string value. If a future doesn't produce a usable value, then the future's type is `Future<void>`.

    - **With an error**: If the asynchronous operation performed by the function fails for any reason, the future completes with an error.

Future (wrong example)

```
void fetchUserOrder() {
  Future.delayed(const  Duration(seconds: 2),
() => print('Large Latte'));
}//fetchUserOrder

void main() {

  print('Fetching user order...’);
  fetchUserOrder();
  print('Done');

}//main
```

The function is doing some asynchronous stuff.

Note: main is an asynchronous function now

Note: ‘Done’ will be print before ‘Large latte’. How to fix this?

Full example in lab_03-dart_101_part_2/04-introducing_futures.dart

Async and Await

- The **`async`** and **`await`** keywords provide a declarative way to define asynchronous functions and use their results. Remember these two basic guidelines when using `async` and `await`:

1. To define an asynchronous function, add **`async`** before the function body and wrap its return type in a `Future`.

2. The **`await`** keyword is used to wait for the result of an asynchronous function before going on and works only inside asynchronous functions.

Fixing the main function

- Let's then fix the main function:
  - First, add the `async` keyword before the function body

```
void main() async {}
```

  - Wrap the return type in a `Future`:

```
Future<void> main() async {}
```

- Now that you have a correctly defined async function, you can use the `await` keyword to wait for a future to complete:

```
await fetchUserOrder();
```

Fixing the fetchUserOrder function

- To fix the `fetchUserOrder` function we can proceed in a similar way
  - First, add the `async` keyword before the function body

    ```
    void fetchUserOrder() async {}
    ```

  - Wrap the return type in a `Future:`

    ```
    Future<void> fetchUserOrder() async {}
    ```

  - Then, `await` the end of the asynchronous operation:

    ```
    await Future.delayed...
    ```

Future (correct example)

```
Future<void> fetchUserOrder() async {
  await Future.delayed(const
      Duration(seconds: 2), () =>
      print('Large Latte'));
}//fetchUserOrder

Future<void> main() async{

  print('Fetching user order...’);
  await fetchUserOrder();
  print('Done');

}//main
```

Note: Now ‘Done’ will be print AFTER ‘Large latte’.

Outline

- Recap
- Classes
- Inheritance
- Other things
- Asynchrony

- **Exercises**
- Homework
- Resources

Exercises

- Exercise 02.01
  - Create a class Vehicle with max_speed, is_moving and mileage instance variables (properly choose the type of the variables). max_speed is constant. is_moving and milage must be properly initiatilized.
  - Create an unnamed constructor with the minimum amount of input arguments.
  - Create also a named constructor Vehicle.used that creates a new Vehicle with a given mileage.
  - Implement two methods start and stop that properly set is_moving
  - Implement also the toString() method of the Vehicle class.
  - Create a method addMiles that takes a named parameter miles, adds that value to the current mileage, and return the new mileage.
  - Properly test the created class capabilities in the main function.

- Exercise 02.02
  - Create a Bus class that extends the Vehicle class and inherit everything from it.
  - Properly inherit the superclass constructors
  - A bus must retain also the current_number_of_passengers and the max_number_of_passengers.
  - Each Bus has a constant max_number_of_passengers equal to 20 and the initial current_number_of_passengers is always 0.
  - Implement a method board that increments the number of passengers by a given value (as much as possible) and return the new number of passengers.
  - Remember to correclty manage the toString() method.
  - Properly test the created class capabilities in the main function.

Exercises

- Exercise 02.03
  - Write an asynchronous function fetchUserRole() that after 3 seconds returns the String ‘admin’. Then, use that function in the main function to print the provided and properly produce the following output:

```
Fetching user role…
The user is an admin.
```

- Exercise 02.04
  - Use the fetchUserRole() function developed in 02.04 to create a new function isAdminUser() that checks if the string provided by fetchUserRole() is ‘admin’ and returns the respective boolean. Use the new function in the main to produce the following output:

```
Checking if user is an admin…
Ok, access granted! (if the user is an admin)
Access denied! (if the user is not an admin)
```

Outline

- Recap
- Classes
- Inheritance
- Other things
- Asynchrony

- Exercises
- **Homework**
- Resources

Homework

- (Try to) Do all the exercises
- Get familiar with Dart 101 (part 1 & 2)
- Get familiar with OOP
- Take a look at Streams https://dart.dev/tutorials/language/streams
- Be sure that the Flutter SDK is working and correctly installed

Outline

- Recap
- Classes
- Inheritance
- Other things
- Asynchrony

- Exercises
- Homework
- **Resources**

Resources

- Code repository of today's lesson and exercises solution
  - https://github.com/gcappon/bwthw/tree/master/lab_03-dart_101_part_2
- Dart language tour
  - https://dart.dev/guides/language/language-tour
- Async and await codelabs
  - https://dart.dev/codelabs/async-await
- Streams tutorial
  - https://dart.dev/tutorials/language/streams