Classes in Dart

- Non-nullable members have to be initialized

- Private members

- Getters, setters, encapsulation

- Constructors with "initializing formal parameters" or with "initializer list"

- Named constructors

- const constructors

- Optional: List as generic type

Reminder: Classes and constructors in Java

```java
package com.journaldev.constructor;

public class Data {

    private String name;
    private int id;

    //no-args constructor
    public Data() {
        this.name = "Default Name";
    }
    //one parameter constructor
    public Data(String n) {
        this.name = n;
    }
    //two parameter constructor
    public Data(String n, int i) {
        this.name = n;
        this.id = i;
    }

    public String getName() {
        return name;
    }
```

```java
    public int getId() {
        return id;
    }

    @Override
    public String toString() {
        return "ID="+id+", Name="+name;
    }
    public static void main(String[] args) {
        Data d = new Data();
        System.out.println(d);

        d = new Data("Java");
        System.out.println(d);

        d = new Data("Pankaj", 25);
        System.out.println(d);

    }
}
```

Copied from https://www.digitalocean.com/community/tutorials/constructor-in-java

Reminder: Inheritance in Java

```java
package com.journaldev.constructor;

public class Person {

    private int age;

    public Person() {
        System.out.println("Person Created");
    }

    public Person(int i) {
        this.age = i;
        System.out.println("Person Created with Age = " + i);
    }

}
```

```java
public class Student extends Person {

    private String name;

    public Student() {
        System.out.println("Student Created");
    }

    public Student(int i, String n) {
        super(i); // super class constructor called
        this.name = n;
        System.out.println("Student Created with name = " + n);
    }

}
```

Copied from https://www.digitalocean.com/community/tutorials/constructor-in-java

Classes in Dart

Class names in Dart should start with capital letters (exceptions: num, int, double).

Non-nullable fields have to be initialized:

```
class User {
  String firstName;
  String lastName;
}
```

→

```
Non-nullable instance field 'firstName' must be initialized.
Try adding an initializer expression, or a generative constructor that initializes it
```

```
void main() {
  User u1 = new User();       // new is not needed in Dart !
  User u2 = User();           // class name on left side can be replaced by 'var'
  var u3 = User();

  print(u3.firstName);        // prints empty line
  u3.firstName = "Franz";
  print(u3.firstName);        // prints "Franz"
  print(u3);                  // prints "Instance of 'User'"
}

class User {
  String firstName = "";
  String lastName = "";
}
```

Class methods

```dart
void main() {
  var u = User();
  u.firstName = "Fritz";
  u.lastName = "Fischer";
  print (u.getFullName());    // prints "Fritz Fischer"
}


class User {
  String firstName = "";
  String lastName = "";

  String getFullName() {
    return firstName + " " + lastName;
  }
}
```

Same method with "arrow syntax":

```dart
String getFullName2() => firstName + " " + lastName;
```

Getter

```dart
void main() {
  var u = User();
  u.firstName = "Fritz";
  u.lastName = "Fischer";
  print(u.getFullName()); // prints "Fritz Fischer"
  print(u.fullName);      // prints the same
}

class User {
  String firstName = "";
  String lastName = "";

  String getFullName() {
    return firstName + " " + lastName;
  }

  String get fullName {
    return firstName + " " + lastName;
  }
}
```

Usage in Flutter (sample from scaffold.dart):

```dart
/// Whether this scaffold has a non-null [Scaffold.appBar].
bool get hasAppBar => widget.appBar != null;
```

Same with "arrow syntax":

```dart
String get fullName => firstName + " " + lastName;
```

Setter

```dart
void main() {
  var u = User();
  u.setFullName("Olaf Scholz");
  print(u.firstName);      // prints Olaf
}

class User {
  String firstName = "";
  String lastName = "";

  void setFullName(String fullName) {
    var names = fullName.split(" ");
    if (names.length == 2) {
      firstName = names[0];
      lastName = names[1];
    }
  }

  String get fullName => firstName + " " + lastName;
}
```

```dart
Run | Debug
void main() {
  var u = User();
  u.fullName = "Olaf Scholz";
  print(u.firstName);          // prints Olaf
}

class User {
  String firstName = "";
  String lastName = "";

  set fullName(String fullName) {
    var names = fullName.split(" ");
    if (names.length == 2) {
      firstName = names[0];
      lastName = names[1];
    }
  }

  String get fullName => firstName + " " + lastName;
}
```

Put classes in their own files

It is a good practice to put class "Abc" into a file "abc.dart":

```dart
lib > main.dart > ...
1
2   import 'user.dart';
3   |
    Run | Debug
4 v void main() {
5       var u = User();
6       u.fullName = "Olaf Scholz";
7       print(u.firstName);              // prints Olaf
8   }
9
```

```dart
lib > user.dart > ...
1
2   class User {
3       String firstName = "";
4       String lastName = "";
5
6       set fullName(String fullName) {
7           var names = fullName.split(" ");
8         if (names.length == 2) {
9               firstName = names[0];
10              lastName = names[1];
11          }
12      }
13
14      String get fullName => firstName + " " + lastName;
15  }
16
```

Private members

Members, whose names start with "_" are private.
They can only be accessed inside the file, where the class is defined:

```dart
// user.dart > ...

class User {
  String _firstName = "";
  String lastName = "";

  set fullName(String fullName) {
        var names = fullName.split(" ");
    if (names.length == 2) {
      _firstName = names[0];
      lastName = names[1];
    }
  }

  String get fullName => _firstName + " " + lastName;
}

String getFirstNameOfUser(User u) {
  return u._firstName;
}
```

```dart
// main.dart > ...

import 'user.dart';

Run | Debug
void main() {
  var u = User();
  u.fullName = "Olaf Scholz";
  print(u._firstName);         // prints Olaf
}

String getFirstNameOfUser(User u) {
  return u._firstName;
}
```

Encapsulation made easy with IntelliSense

```dart
class User {
  String firstName = "";
  String lastName = "";
```

Set cursor on "firstName", press right mouse button and select:

```
Format Document With…
Refactor…                    Ctrl+Shift+R
Source Action
```

```dart
class User {
  String firstName = "";
  String
        More Actions…
         💡 Remove type annotation
  set ful
         💡 Encapsulate field
        v
```

```dart
class User {
  String _firstName = "";

  String get firstName => _firstName;

  set firstName(String value) {
    _firstName = value;
  }

  String lastName = "";
```

main.dart is unchanged:

```dart
void main() {
  var u = User();
  u.fullName = "Olaf Scholz";
  print(u.firstName);
}
```

Advantages of a setter:
- it can check if the value makes sense (e.g. in our case if value has at least 2 characters)
- you can set a breakpoint in it

Sample of encapsulation in Flutter sources


```dart
class _RenderColoredBox extends RenderProxyBoxWithHitTestBehavior {
  _RenderColoredBox({ required Color color })
    : _color = color,
      super(behavior: HitTestBehavior.opaque);

  /// The fill color for this render object.
  ///
  /// This parameter must not be null.
  Color get color => _color;
  Color _color;
  set color(Color value) {
    if (value == _color) {
      return;
    }
    _color = value;
    markNeedsPaint();
  }
}
```

Classes: constructors (part 1)

Remember the issue we had at the beginning:

```
class User {
  String firstName;
  String lastName = "";
```

```
Non-nullable instance field 'firstName' must be initialized.
Try adding an initializer expression, or a generative constructor that initializes it
```

We solved it with an initializer expression:

```
String firstName = "";
```

As mentioned in the error text above, an alternative is to define a constructor
(in short: c-tor):

```
class User {
  String firstName;
  String lastName;

  User (this.firstName, this.lastName);
```

```
void main() {
  //var u = User();     // error: 2 positional arguments expected
  var u = User("Fritz", "Fischer");
  print(u.firstName);                    // prints Fritz
}
```

Classes: constructors (part 2)

The constructor and other class methods have several significant differences.

1. The constructor has the same name as the class name.
2. The constructor does not have a return type.
3. When an object is created, the constructor is automatically invoked.
4. If no constructor is specified, the default no-argument constructor is used.

There are 2 types of constructors:

```dart
class User {
  String firstName;
  String lastName;

  // c-tor with "initializing formal parameters"
  User (this.firstName, this.lastName);
```

Most frequently used

```dart
class User {
  String firstName;
  String lastName;

  // c-tor with "initializer list"
  User(String fName, String lName) : firstName = fName, lastName = lName;
```

More seldom used

C-tors can have bodies

```
int numberOfUsers = 0;

class User {
  String firstName;
  String lastName;

  User(this.firstName, this.lastName) {
    numberOfUsers++;
  }
}
```

Better than above with a global variable: use a static variable in the class.
Static class variables exist only once for all class objects !

```
class User {
  static int numberOfUsers = 0;

  String firstName;
  String lastName;

  User(this.firstName, this.lastName) {
    numberOfUsers++;
  }
}
```

```
void main() {
  var u = User("Fritz", "Fischer");
  var u2 = User("Olaf", "Scholz");
  print(User.numberOfUsers);        // prints 2
}
```

Beside the default c-tor, classes can have "named c-tors"

```dart
class User {
  String firstName;
  String lastName;

  User(this.firstName, this.lastName);

  User.withFullName(String fullName) : firstName = "", lastName = "" {
    this.fullName = fullName;
  }
}
```

```dart
void main() {
  var u = User("Fritz", "Fischer");
  var u2 = User.withFullName("Olaf Scholz");
  print("${u2.lastName} ${u2.firstName}");
}
```

Remark: in C# a class can have several c-tors with the same name,
as long as their argument-list is different.

Strange: Initializing fields in the body is not enough

```
User.withFullName(String fullName) {
    firstName = "";
    lastName = "";
    this.fullName = fullName;
}
```

```
Non-nullable instance field 'firstName' must be initialized.
Try adding an initializer expression, or add a field initializer in this constructor, or mark it
'late'. dart(not_initialized_non_nullable_instance_field)

Non-nullable instance field 'lastName' must be initialized.
Try adding an initializer expression, or add a field initializer in this constructor, or mark it
'late'. dart(not_initialized_non_nullable_instance_field)
```

Samples of using named c-tors in Flutter

Taken from our hello_world app:

```
Image.network(
    "https://fdg-ab.de/wp-content/uploads/2021/03/logo_fdg_neu_freigestellt.png",
    width: 150), // Image.network

ClipRRect(
    borderRadius: BorderRadius.all(Radius.circular(30)),
    // next 2 lines would round only top-left and bottom-right corners
    //   borderRadius: BorderRadius.only(
    //   topLeft: Radius.circular(30), bottomRight: Radius.circular(30)),
    child: Image.asset("assets/images/snoopy_christmas.jpg",
        width: 250)), // Image.asset // ClipRRect
```

Sample of a definition of named c-tors in Flutter

```
/// A radius for either circular or elliptical shapes.
class Radius {
  /// Constructs a circular radius. [x] and [y] will have the same radius value.
  ///
  /// ![](https://flutter.github.io/assets-for-api-docs/assets/dart-ui/radius_circula
  /// ![](https://flutter.github.io/assets-for-api-docs/assets/dart-ui/radius_circula
  const Radius.circular(double radius) : this.elliptical(radius, radius);

  /// Constructs an elliptical radius with the given radii.
  ///
  /// ![](https://flutter.github.io/assets-for-api-docs/assets/dart-ui/radius_ellipti
  /// ![](https://flutter.github.io/assets-for-api-docs/assets/dart-ui/radius_ellipti
  const Radius.elliptical(this.x, this.y);

  /// The radius value on the horizontal axis.
  final double x;

  /// The radius value on the vertical axis.
  final double y;
```
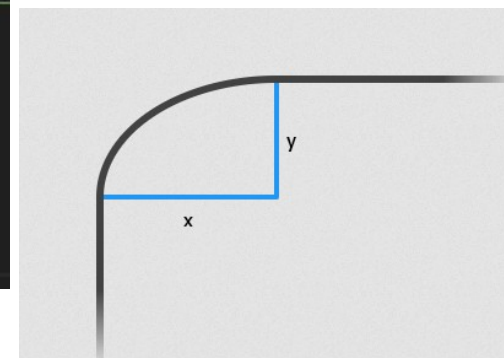
The keywords "final" and "const" inside classes are explained in the next slides !

Exercise

Create a class Rectangle in an own file rectangle.dart inside directoy lib.

Create the class in such a way that you can use it in main.dart in the following way:

```
(new) Rectangle Rectangle(double width, double height)
lib/rectangle.dart
var rect = Rectangle(200, 50);
print("calling method getArea: ${rect.getArea()}");
print("calling setter area: ${rect.area}");
```

Possible solution:

```
class Rectangle {
  Rectangle(this.width, this.height);
  double width;
  double height;

  double getArea() {
    return width * height;
  }

  double get area => width * height ;
}
```

Final members

In our last Powerpoint "dart_basics" we already discussed final variables inside a method.

Final members of a class must be initialized either by value or in a c-tor.
Once initialized, their value cannot be changed.

```dart
class Rectangle {
  Rectangle(this.width, this.height);
  final double width;
  final double height;

  void modifyInClass() {
    width = 23;              // error: 'width' can't be used as a setter because it's final.
  }
}

void modifyOutsideClass() {
  var rect = Rectangle(10, 20);
  rect.width = 23;         // same error as above
}
```

Const constructors

A class can have a const constructor, when all its members are final.

```dart
class Rectangle {
  const Rectangle(this.width, this.height);

  final double width;
  final double height;
}
```

When 2 objects are created with a const c-tor and both have the same compile-time arguments in the c-tor, only one object is created for both:

```dart
void testConstConstructor() {
  var rect1 = const Rectangle(10, 20);
  var rect2 = const Rectangle(11, 20);
  var rect3 = const Rectangle(10, 20);

  print("${rect1.hashCode}, ${rect2.hashCode}, ${rect3.hashCode}");
}
```

605301925, 931157535, 605301925

Const constructors need const arguments

```
Arguments of a constant creation must be constant expressions.
Try making the argument a valid constant, or use 'new' to call the
constructor. dart(const_with_non_constant_argument)

double myWidth

Type: double

View Problem (Alt+F8)    No quick fixes available

double myWidth  = 300;
var rect4 = const Rectangle(myWidth, 50);
```

Solutions:
a) use const argument   b) use "normal" and not the const c-tor

```
const double myWidth  = 300;
var rect4 = const Rectangle(myWidth, 50);
```

```
double myWidth  = 300;
var rect4 = Rectangle(myWidth, 50);
```

Const constructors in Flutter

In flutter we saw warnings like

```
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      const Text("first line"),

      SizedBox(height: 20),          Use 'const' with the constructor to improve performance.
                                     Try adding the 'const' keyword to the constructor invocation. dart(prefer_const_constructors)

      const Text("second line"),
```

```
class SizedBox extends SingleChildRenderObjectWidget {
  /// Creates a fixed size box. The [width] and [height] parameters can be null
  /// to indicate that the size of the box should not be constrained in
  /// the corresponding dimension.
  const SizedBox({ super.key, this.width, this.height, super.child });

  /// Creates a box whose [width] and [height] are equal.
  const SizedBox.square({super.key, super.child, double? dimension})
    : width = dimension,
      height = dimension;

  /// If non-null, requires the child to have exactly this width.
  final double? width;

  /// If non-null, requires the child to have exactly this height.
  final double? height;
```

Inheritance

When base class has a c-tor with arguments, derived class must call it:

```
class PayingUser extends User {

  String iban="unknown";

}
```

```
The superclass 'User' doesn't have a zero argument constructor.
Try declaring a zero argument constructor in 'User', or declaring a constructor in
PayingUser that explicitly invokes a constructor in
'User'. dart(no_default_super_constructor)
```

```
class PayingUser extends User {
  PayingUser(String fName, String lName, String ib) :  iban = ib, super (fName, lName);

  String iban;
}
```

3 possible c-tors for derived classes

With initializer list and calling base class c-tor:

```
PayingUser(String fName, String lName, String ib) :  iban = ib, super (fName, lName);
```

With initializing formal parameters and calling base class c-tor:

```
PayingUser(String fName, String lName, this.iban) : super(fName, lName);
```

Only with initializing formal parameters:

```
PayingUser(super.fName, super.lName, this.iban);
```

25

Super class methods cannot be called by base class objects

```dart
class PayingUser extends User {
  PayingUser(String fName, String lName, this.iban) : super(fName, lName);

  String iban;

  void debit() {
    print("debit called for user $fullName");
  }
}
```

```dart
User userX = User("firstX", "lastX");  // same as    var userX = User("firstX", "lastX");
PayingUser userY = PayingUser("firstY", "lastY", "ibanY");  // also possible with var
User userZ = PayingUser("firstZ", "lastZ", "ibanZ");

userY.debit();
userZ.debit();
```

The method 'debit' isn't defined for the type 'User'.
Try correcting the name to the name of an existing method, or defining a method named
'debit'. dart(undefined_method)

```dart
if (userZ is PayingUser) {
  userZ.debit();
}
```

26

Overriding method toString() inherited from Object

Without overriding:

```
User userX = User("firstX", "lastX");
print(userX);  // same as print(userX.ToString());
```

→ `Instance of 'User'`

In class User:

```
@override
String toString() => "User $firstName $lastName";
```

The line "@override" is optional.

In derived class PayingUser:

```
@override
String toString() => "PayingUser $firstName $lastName $iban";
```

```
User userX = User("firstX", "lastX");  // same as    var userX = User("firstX", "lastX");
PayingUser userY = PayingUser("firstY", "lastY", "ibanY");  // also possible with var
User userZ = PayingUser("firstZ", "lastZ", "ibanZ");

print(userX);      User firstX lastX
print(userY);      PayingUser firstY lastY ibanY
print(userZ);      PayingUser firstZ lastZ ibanZ
```

List as a sample of a generic type

We used already one in our first hello.dart:

```dart
void main(List<String> args) {
  for (var arg in args) {
    print(arg);
  }
}
```

Other samples:

```dart
void testList() {
  List<int> intList = [1, 2, 3, 5, 7, 11, 13];
  List<User> userList = [User.withFullName("Franz Maier")];

  print("last in intList is ${intList.last}");
  print("intList is $intList");

  userList.add(PayingUser("Willi", "Zahn", "ibanWZ"));
  for (var user in userList) {
    print("$user");
  }
  userList.removeLast();
}
```

```
last in intList is 13
intList is [1, 2, 3, 5, 7, 11, 13]
```

```
User Franz Maier
PayingUser Willi Zahn ibanWZ
```

28

Compare Dart and C# e.g. for using DateTime

```dart
main.dart  ×

main.dart > ...
      Run | Debug
1   void main(List<String> args) {
2     var dt = DateTime(1950, 12, 20);
3     var today = DateTime.now();
4     var timespan = today.difference(dt);
5     var age = timespan.inDays / 365.25;
6     print("age is $age");
7   }
```

```csharp
Program.cs  ×
TestDateTime                           Program

class Program
{
    static void Main(string[] args)
    {
        var dt = new DateTime(1950, 12, 20);
        var today = DateTime.Now;
        var timespan = today - dt;
        var age = timespan.TotalDays / 365.25;
        Console.WriteLine($"age is {age}");
    }
}
```

Not yet discussed

- Exceptions

- Maps (also called Dictionaries)

- Type dynamic

- Mixins

- Asynchronous programming with async / await

- Operator overloading

- Extension methods

- …

Useful Links

- Dart "Spickzettel":
  https://dart.dev/codelabs/dart-cheatsheet

- Dart Intro in about one hour:
  https://www.youtube.com/watch?v=JZukfxvc7Mc

  For more tutorials look in YouTube for "dart programming tutorial".
  Don't forget "programming", otherwise you might end up with