Classes in Dart

- Non-nullable members have to be initialized
- Class methods, getters, setters
- Private members
- Constructors with “initializing formal parameters” or with “initializer list”
- Named constructors and const constructors
- Derived classes and overridden methods
- 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;
	}
	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

Classes in Dart

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

Difference to Java: 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
```

Solution 1:
initialize the fields:

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

Solution 2:
use nullable types:

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

```
The operator '+' can't be unconditionally invoked
because the receiver can be 'null'.
```

Disadvantage:

```
var fullName = firstName + " " + lastName;
```

“new” is not needed in Dart

```dart
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

```
Run | Debug
void main() {
  User u = new User();
  u.firstName = "Max";
  u.lastName = "Mustermann";
  print(u.getFullName());          // prints "Max Mustermann"
  u.setFullName("Fritz Fischer");
  print(u.getFullName());          // prints "Fritz Fischer"
}

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

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

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

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

```
void main() {
  User u = new User();
  u.setFullName("Fritz Fischer");
  print(u.fullName);          // prints "Fritz Fischer"
  u.fullName = "Max Mustermann";
}     There isn't a setter named 'fullName' in class 'User'.

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

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

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

```
void main() {
  User u = new User();
  u.fullName = "Fritz Fischer";
  print(u.fullName);          // prints "Fritz Fischer"
}

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

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

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

With Getter and Setter, it looks as if class User has a field “fullName”.

Put classes in their own files

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

lib > main.dart > ...

```dart
import 'user.dart';

Run | Debug
void main() {
  User u = new User();
  u.setFullName("Fritz Fischer");
  print(u.firstName);    // prints "Fritz"
}
```

lib > user.dart > ...

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

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

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

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:

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

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

Most frequently used

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

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

More seldom used

Text was copied from https://www.educative.io/answers/what-are-constructors-in-dart-programming

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
}
```

Comparison with Java

```java
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;
        }
```

```dart
class Data {
  String _name;

  Data(String n) {
    _name = n;
  }
```

```
Non-nullable instance field '_name' must be initialized.
```

```dart
Data(String n) : _name = n;
```

Initializer list

```dart
Data(this._name);
```

Initializing formal parameter

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

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

  User(this.firstName, this.lastName);

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

```
Run | Debug
void main() {
  User u1 = new User("Fritz", "Fischer");
  User u2 = new User.withFullName("Hans Müller");
}
```

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

Again: Initializing fields in the body is not enough

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

Same as described in slide 13:

```
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)
```

Dart wants an initializer list as used on last page:

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

Samples of using named c-tors in Flutter

Taken from our images 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
```

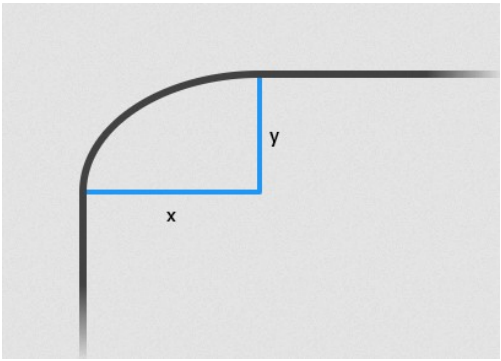
Sample of a definition of named c-tors in Flutter

```dart
/// 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”.

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("area of rect is ${rect.getArea()}");
```

“Bonus” task: Ensure that width and height cannot be set to negative values.

Hint: use the body of Rectangle‘s constructor to not accept negative values.

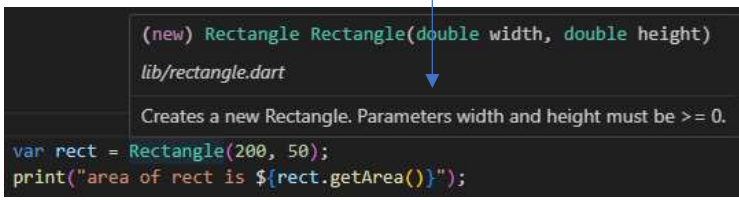
Possible solution

```dart
class Rectangle {
  double width;
  double height;

  /// Creates a new Rectangle. Parameters width and height must be >= 0.
  Rectangle (this.width, this.height) {
    if (width < 0) {
      width = 0;
    }
    if (height < 0) {
      height = 0;
    }
  }

  double getArea() {
    return width * height;
  }
}
```

This comment starting with "///"
is shown in main.dart by Intellisense:



Alternative: throw an exception:

```dart
if (width < 0) {
  throw("negative width is not allowed");
}
```

Private members

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

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

  String getFullName() {
    return "$_firstName $lastName";
  }

  String get firstName {
    return _firstName;
  }
}

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

```dart
import 'user.dart';

Run | Debug
void main() {
  User u = new User();
  print(u._firstName);
  print(u.firstName);
  u.firstName = "Otto";
}

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

There isn't a setter named 'firstName' in class 'User'.

To allow read access, define a getter.

Private member encapsulated with Getter and Setter

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

  String get firstName {
    return _firstName;
  }

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

```dart
import 'user.dart';

Run | Debug
void main() {
  User u = new User();
  print(u._firstName);
  print(u.firstName);
  u.firstName = "Otto";
}
```

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

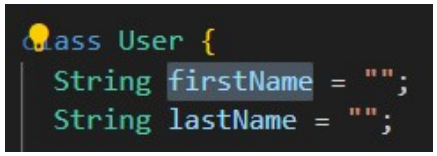
Sample of such an encapsulation in Flutter sources

basic.dart

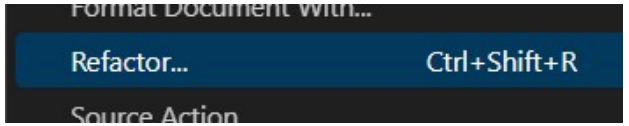
C: > FlutterSDK > flutter_windows_3.3.10-stable > flutter > packages > flutter > lib > src > widgets > basic.dart > _RenderColoredBox

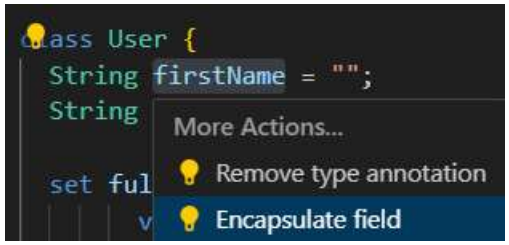
```
7727
7728  class _RenderColoredBox extends RenderProxyBoxWithHitTestBehavior {
7729    _RenderColoredBox({ required Color color })
7730      : _color = color,
7731        super(behavior: HitTestBehavior.opaque);
7732
7733    /// The fill color for this render object.
7734    ///
7735    /// This parameter must not be null.
7736    Color get color => _color;
7737    Color _color;
7738    set color(Color value) {
7739      if (value == _color) {
7740        return;
7741      }
7742      _color = value;
7743      markNeedsPaint();
7744    }
7745
```

Encapsulation made easy with IntelliSense



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





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

  String get firstName => _firstName;

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

  String lastName = "";
```

Access in main.dart is unchanged:

```
void main() {
  User u = new User();
  u.firstName = "Max";
  print(u.firstName);
```

Final members

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

Final members of a class are usually initialized in a c-tor or by setting an initial value. 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.

```
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:

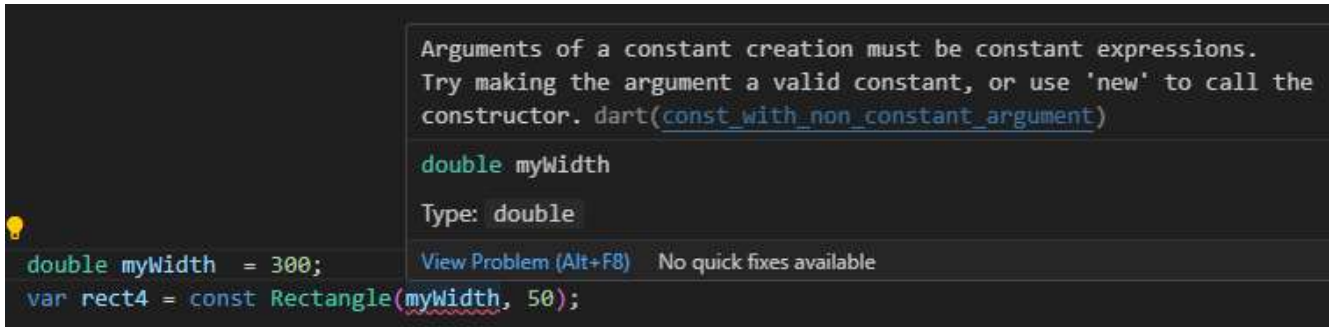
```
void testConstConstructor() {
  var rect1 = const Rectangle(10, 20);
  var rect2 = const Rectangle(11, 20);
  var rect3 = const Rectangle(10, 20);

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

→

```
605301925, 931157535, 605301925
```

Const constructors need const arguments



Solutions:

a) use const argument

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

b) use “normal” and not the const c-tor

```
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),

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

```
Use 'const' with the constructor to improve performance.
Try adding the 'const' keyword to the constructor invocation. dart(prefer_const_constructors)
```

```
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;
```

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

Inheritance

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

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

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

```dart
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)
```

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

  String iban;
}
```

Java reminder:

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

3 possible c-tors for derived classes

With initializer list and calling base class c-tor:

Super call must be last in initializer list.

```
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);
```

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() {
  return "User $firstName $lastName";
}
```

Java Reminder:

```
@Override
public String toString() {
        return "ID="+id+", Name="+name;
}
```

In derived class PayingUser:

```
@override
String toString() {
  return "PayingUser $firstName $lastName $iban";
}
```

BTW: In Dart the line “@override” is optional.

Using overridden methods in main.dart

```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");

print(userX);
print(userY);
print(userZ);
```

```
User firstX lastX
PayingUser firstY lastY ibanY
PayingUser firstZ lastZ ibanZ
```

```dart
List<User> users = [userX, userY, userZ];

for (var user in users) {
  print(user);
}
```

```
User firstX lastX
PayingUser firstY lastY ibanY
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
```

Collection if

```
Run | Debug
void main(List<String> args) {
  print (getList(useThree: true));
  print (getList(useThree: false));
}

List<int> getList ({required bool useThree}) {
  var list = [1, 2, if(useThree) 3, 4, 5];
  return list;
}
```

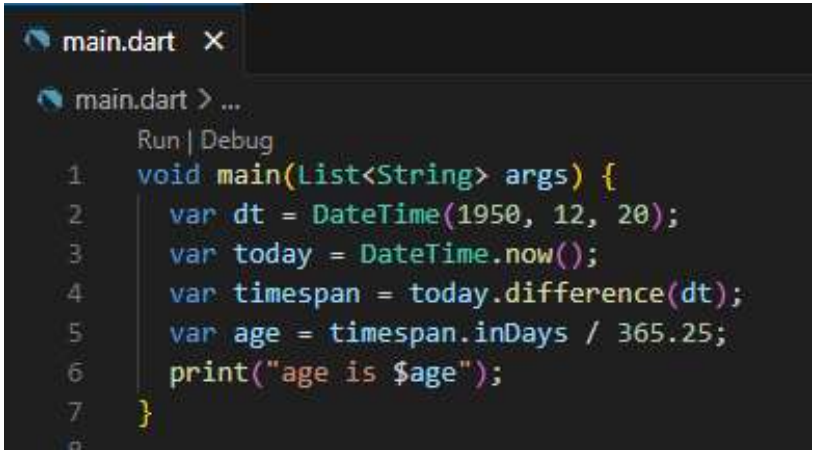
Output:

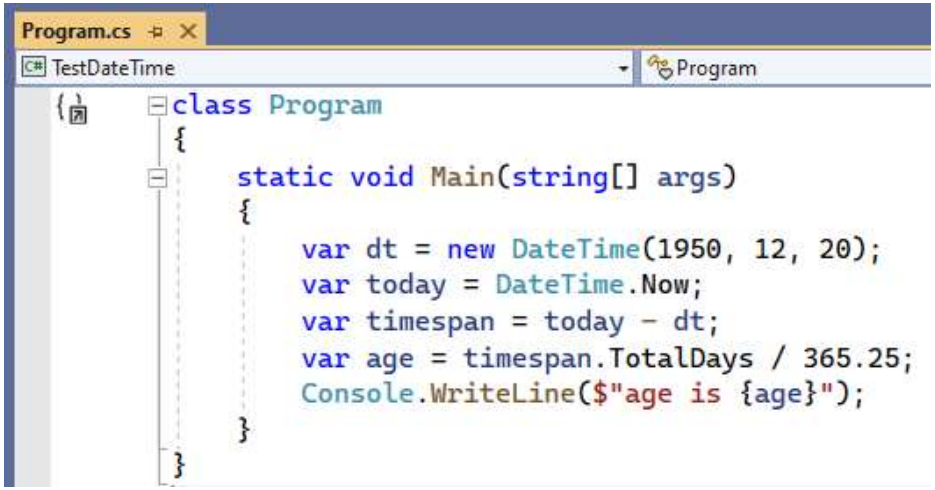
```
[1, 2, 3, 4, 5]
[1, 2, 4, 5]
```

You cannot use “if … else …” here, but you can use a conditional expression:

```
var list = [1, 2, useThree ?  3 : 33, 4, 5];
```

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





Not yet discussed

- maps (key/value pairs, also called dictionaries)
- type dynamic
- mixins
- catching exceptions
- 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