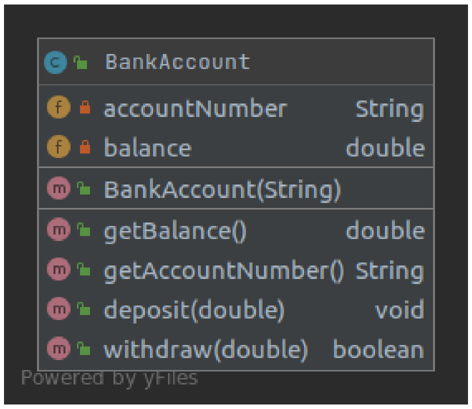
# OOP
# Lab 2.

## 🎯 Objectives

- Create simple classes
    - Attributes (data)
    - Methods (behavior)
        - Constructors
        - `toString()` method
- Create instances and perform operations on them
- The `this` reference

## Exercise 1.

Create a `BankAccount` class.



Data (attributes):

- `balance` - an item of type double
- `accountNumber` - an item of type String

| **Method** | **Description** | **Inputs** | **Output** |
|---|---|---|---|
| `BankAccount` | Constructor: initialisation of the object (accountNumber, balance) | A String object (accountNumber) | Not applicable |

| | | | |
|---|---|---|---|
| `getAccountNumber` | Returns the account number | None | An item of type String |
| `getBalance` | Returns the balance | None | An item of type double |
| `deposit` | Accepts an item of type double and adds it to the balance. Only positive amounts are added to the balance! | An item of type double | None |
| `withdraw` | Accepts an item of type double and checks if there are sufficient funds to make a withdrawal. If there are not, returns false. Otherwise, subtracts the amount from the balance and returns true. | An item of type double | An item of type boolean. |

Test your class (Main class - main method)

1. Create a bank account (accountNumber: OTP00001)

```
BankAccount account1 = new BankAccount("OTP00001");
```

2. Print the bank account number and the balance

```
System.out.println(account1.getAccountNumber()+": "+account1.getBalance());
```

3. Deposit 1000 EUR

```
account1.deposit(1000);
```

4. Print the bank account number and the balance
5. Withdraw 500 EUR

```
boolean result = account1.withdraw(500);
if ( !result ){
   System.out.println("You do not have sufficient funds for this operation!");
}
```

6. Print the bank account number and the balance

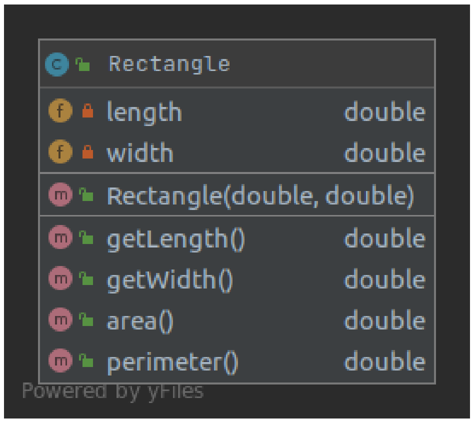
7. Withdraw 1000 EUR
8. Print the bank account number and the balance
9. Create a second bank account (accountNumber: OTP00002)
10. Print the bank account number and the balance
11. Deposit 2000 EUR
12. Print the bank account number and the balance

In the case of a withdrawal, always check the result of the operation. In case of insufficient funds, always print the reason for failure.

## Exercise 2.

Create a class `Rectangle`.

- A rectangle is a shape described by a `length` and a `width`, both attributes are real numbers.
- You should be able to initialize the attributes of a rectangle with two positive real numbers (constructor).
- You should be able to calculate the area and the perimeter of a rectangle (`area()` and `perimeter()` methods).
- Create getter methods for both attributes (`getLength(), getWidth()`).



**Test your class (Main class - main method):**

1. Create an array of 10 references of type Rectangle.

```
Rectangle[] rectangles = new Rectangle[ 10 ];
```

2. Initialize each element of the array with a new rectangle. Generate randomly the value of the `length` and `width` attributes (`1 <= length <= 10,  1 <= width <= 10`).

```
// use a random generator
Random rand = new Random();

//generate positive random numbers less than a bound
double length = 1 + rand.nextInt(10);
double width = 1 + rand.nextInt(10) ;
rectangles[ i ] = new Rectangle(length, width);
```

3. Print the rectangles to the standard output. Print the following about a rectangle: length, width, perimeter, area.
4. Calculate the total area of the generated rectangles.

## Exercise 3.

a. Create a `DateUtil` class. This class defines two utility methods for dates. Both functions should be declared `public static`.

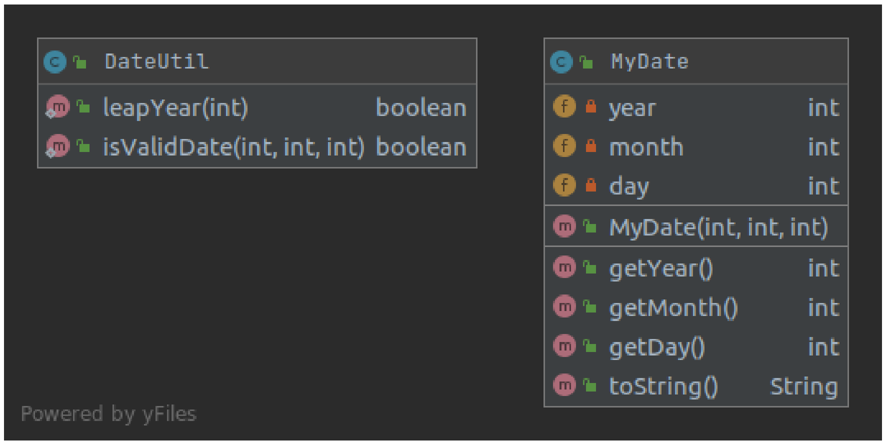
- The `leapYear` method checks whether its parameter is a leap year.

```
public static boolean leapYear(int year)
```

How to know if it is a leap year? https://www.mathsisfun.com/leap-years.html

- The `isValid` method checks whether its parameters (`year, month, day`) form a valid date.

```
public static boolean isValidDate(int year, int month, int day)
```



| DateUtil | |
|---|---|
| leapYear(int) | boolean |
| isValidDate(int, int, int) | boolean |

| MyDate | |
|---|---|
| year | int |
| month | int |
| day | int |
| MyDate(int, int, int) | |
| getYear() | int |
| getMonth() | int |
| getDay() | int |
| toString() | String |

Test the class (Main class, main method)!
Run the following code. You should get only true on the output!

```
System.out.println(DateUtil.isValidDate(2000,2, 29) == true);
System.out.println(DateUtil.isValidDate(2000,2, 30) == false);
System.out.println(DateUtil.isValidDate(1900,2, 29) == false);
System.out.println(DateUtil.isValidDate(1900,2, 28) == true);
System.out.println(DateUtil.isValidDate(-1900,2, 28) == false);
System.out.println(DateUtil.isValidDate(0,2, 28) == false);
System.out.println(DateUtil.isValidDate(2021,2, 29) == false);
System.out.println(DateUtil.isValidDate(2020,2, 29) == true);
System.out.println(DateUtil.isValidDate(2020,1, 32) == false);
System.out.println(DateUtil.isValidDate(2020,1, 0) == false);
System.out.println(DateUtil.isValidDate(2020,0, 0) == false);
System.out.println(DateUtil.isValidDate(2020,4, 31) == false);
System.out.println(DateUtil.isValidDate(2020,1, 31) == true);
```

b. Create a `MyDate` class.
**Attributes:** `year, month, day` are integer values and should form a valid date.
**Methods:**

- Constructor (3 parameters)
  Initializes the attributes if and only if the parameters form a valid date.
- Create a getter method for each attribute.
- Create a `toString` method which returns the date object in a textual format.

```
public String toString()
```

No setter methods. The instances of the `MyDate` class will be **immutable (constants)**.

# OOP
# Lab 2.

Test the `MyDate` class!

Create a `main` method which uses the random number generator (class `Random` from the `java.util` package) and generates 1000 random dates and prints them to the standard output.

- You should print only the valid dates.
- Count the number of invalid dates generated and print it to the standard output!