OOP
Lab 6.

🎯 Objectives

- Working with **`static`** members
  - Static attributes
  - Static methods
- Working with constants (**`static final`**)
- Using blank `final` attributes (**`final`**)

Create a Project/Module with 2 packages:

- `lab6_1`
- `lab6_2`

Each exercise has its own package and Main class (main method).

Exercise 1.

Structure:

- `lab6_1`
  - `Main`
  - `BankAccount`
  - `Customer`
  - `Bank`

a. `BankAccount` class

In this exercise you have to modify the BankAccount class in order to generate the accountNumber. This number should be unique!

- Copy the `BankAccount` class from the previous week and modify according to the diagram shown in Fig. 1:
- Modify the attribute accountNumber (Add the `final` modifier!)

```
private final String accountNumber;
```

- Add a String **constant** to the class to represent the prefix of each account:

```
public static final String PREFIX = "OTP";
```

OOP
Lab 6.

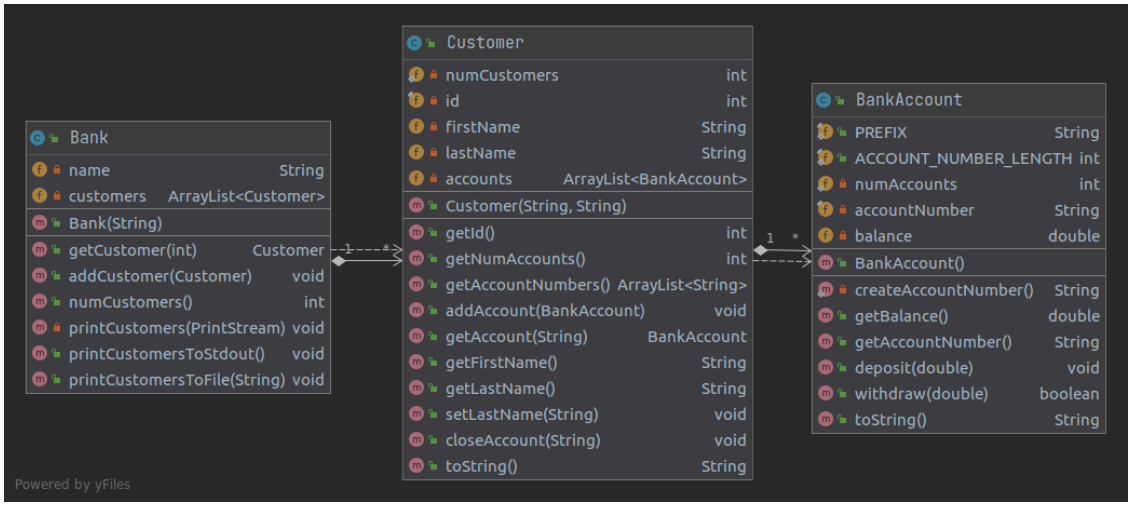

Fig. 1. Class diagram Exercise 1

- You will need an extra **static field** to count the number of created accounts.

```
private static int numAccounts = 0;
```

- You will need another **constant** to represent the length (number of characters) of an account number.

```
public static final int ACCOUNT_NUMBER_LENGTH = 10;
```

- Create a **private method** (*helper method*) that generates the correct `accountNumber` for the current object. For example: the 1st accountNumber should be: “OTP0000001”, the 2nd “OTP0000002”, and the 100th “OTP0000100”

```
private static String createAccountNumber()
```

- Modify the constructor of the class:

```
public BankAccount() {
   ++numAccounts;
   this.accountNumber = createAccountNumber();
}
```

- Test the `BankAccount` class (`main` method):
  - Create an `ArrayList` of BankAccounts.
  - Add 150 accounts to the ArrayList and print it to the standard output.

b. `Customer` class

You should modify the Customer class (from `lab4_2`) to include a unique id for each customer.

- Copy the `Customer.java` file from `lab4_2` to `lab6_1` package.
- Add a blank `final id` attribute and initialize it properly in the constructor. The `id` should have values starting from 1.
- Generate a getter method for the `id` attribute.
- Create a `getAccountNumbers` which returns a list containing the account numbers of the current customer.
- Modify the `toString` method in order to include the value of the `id` attribute.
- Test the `Customer` class (main method): create an ArrayList of customers, add 3 customers each having at least one account, then print it to the standard output.

c. `Bank` class

- Create a Bank class. Each bank has a name and a list of customers. Implement the class according to the diagram shown in Fig. 1.
- Create a getCustomer method that returns the customer having the `id` equal to `customerId`

```
public Customer getCustomer( int customerId )
```

- Create a general `printCustomers` method which prints the customers to a stream. This method is **private**.

```
private void printCustomers( PrintStream ps ) {
   ps.println("ID, Firstname, Last name, Number of bank accounts");
   for( Customer customer: customers ) {
       ps.println( customer.getId()+", " + customer.getFirstName() + ", "+
                   customer.getLastName()+", "+customer.getNumAccounts());
   }
   ps.close();
}
```

- You will be able to call these two methods from outside of the class!

```
public void printCustomersToStdout() {
   printCustomers( System.out );
}

public void printCustomersToFile( String filename ) {
   try {
       printCustomers( new PrintStream(filename ));
   } catch (Exception e) {
       e.printStackTrace();
   }
}
```

- Test the Bank class (main method):
    - Create a bank with the name OTP
    - Add two customers to the bank
    - Add two accounts to each customer
    - Print the customer having id equal to 1 to the standard output
    - Deposit some amount of money in each of the accounts. Get the customer from the bank object, then get the account from the customer object.
    - Print the customer having id equal to 2 to the standard output
    - Print the customers of the bank to a file (e.g. `bank_customers.csv`)

Exercise 2.

Create a `Matrix` class and implement it according to the diagram shown in Fig.2.
Make sure the copy constructors make deep copies of their parameters!

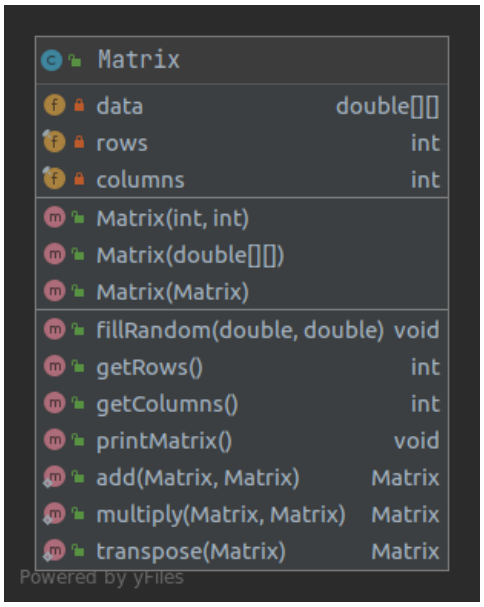

Fig. 2. Matrix class

- Test your class (`Main` class, `main` method)! You may use the following code snippets.

```
double[][] d = { { 1, 2, 3 }, { 4, 5, 6 }, { 9, 1, 3} };
Matrix m0 = new Matrix(d);

System.out.println("m0 rows: " + m0.getRows()+", cols:  " + m0.getColumns());
System.out.println("m0: ");
m0.printMatrix();
System.out.println();

Matrix m00 = new Matrix( m0 );
System.out.println("m00 is a deep copy of m0: ");
m00.printMatrix();
System.out.println("**********************");
```

OOP
Lab 6.


```
Matrix m1 = new Matrix(2, 3);
m1.fillRandom(1,2);
System.out.println("m1: ");
m1.printMatrix();
System.out.println();

Matrix m2 = new Matrix(2, 3);
m2.fillRandom(1,2);
System.out.println("m2: ");
m2.printMatrix();
System.out.println("Sum: m1 + m2");
Matrix.add(m1, m2).printMatrix();

Matrix m3 = new Matrix(3, 4);
m3.fillRandom(0, 1);
System.out.println("m3: ");
m3.printMatrix();

System.out.println("Product: m1 * m3");
Matrix.multiply(m1, m3).printMatrix();

System.out.println("Transpose(m3)");
Matrix.transpose( m3 ).printMatrix();
```