OOP
Lab 3.

🎯 Objectives

- Create associations between classes
- Implement **one-to-one** associations
- Implement **one-to-many** associations
- Arrays of references/objects

Create a Project with 2 packages:

- `lab3_1`
- `lab3_2`

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

Excercise 1.

Structure:

- `lab3_1`
  - `BankAccount`
  - `Customer`
  - `Main`

A. Create a `BankAccount` class (see Fig. 1). You can use the BankAccount class created in Lab 2, but you should add a `toString()` method which returns the state of the object.

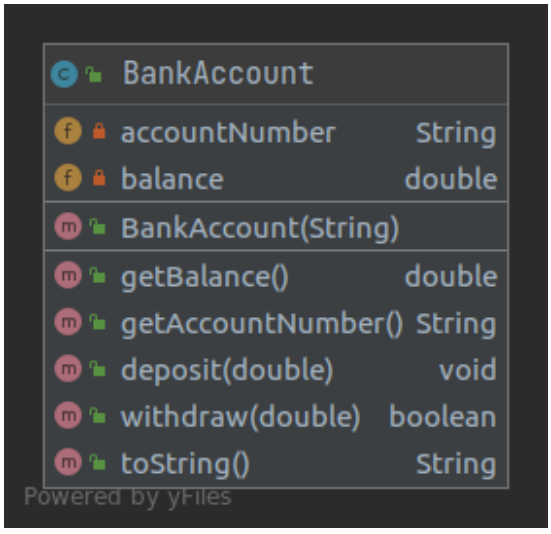

Fig.1. BankAccount class.

B. Create a class `Customer`.

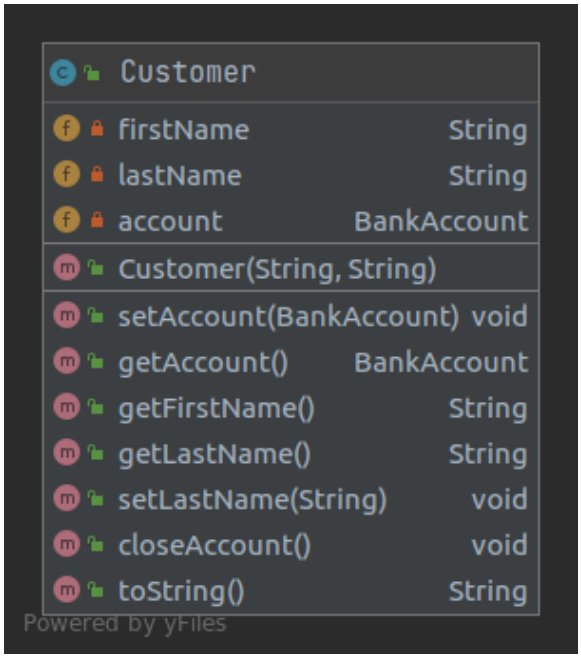

Fig. 2. Customer class.

1. Each customer of a bank is characterized by `firstName, lastName` (family name) and may have a bank account.
2. The constructor of the class takes two parameters, the first and the last name and initializes the corresponding attributes.
3. Each customer may have a single bank account and this will be set later by the `setAccount` method.
4. Create a `getAccount` method, which returns the attached account.
5. Add a `closeAccount` method which sets the account attribute to `null`.
6. Create a `toString` method, which returns the string representation of a customer.

C. Test your classes! (Main class - main method)

- Create a customer with the name John BLACK.

```
Customer customer1 = new Customer("John", "BLACK");
```

- Print the customer using its toString method.

```
System.out.println(customer1.toString());
System.out.println(customer1); // RECOMMENDED APPROACH!!
```

- Set the account of this customer to an account with accountNumber OTP00001.
- Print the customer to the standard output.
- Deposit 1000 EUR in the customer bank account.

```
customer1.getAccount().deposit(1000);
System.out.println(customer1);
```

- Perform other operations with the customer account.
- Create another customer with the name Mary WHITE.
- Set the account of Mary to an account with accountNumber OTP00002.
- Print Mary's data to the standard output.
- Perform some operations with Mary's account.
- Close Mary's account.
- Print Mary's data to the standard output.
- John decides to marry Mary and he wants to share his bank account with Mary. Help John in doing so.
- Print John's and Mary's data to the standard output.

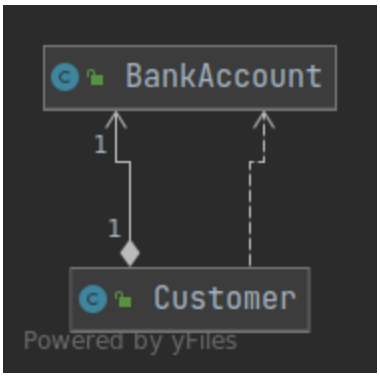

Fig.3. Class diagram. Association. One-to-one relationship. Each Customer has a BankAccount.

Exercise 2.

Structure:

- `lab3_2`
  - `Customer`
  - `Main`

Copy the `Customer` class from `lab3_1` package into `lab3_2` package.

- Modify the `Customer` class in order to permit a customer to have more than one account. We have to change the attribute implementing the relationship between `Customer` and `BankAccount`. This time we have a **one-to-many** relationship: one customer may have many accounts.

Modify the attribute account:

```
// constant
public static final int MAX_ACCOUNTS = 10;

// number of accounts
private int numAccounts;
// an array for the accounts
private BankAccount accounts[] = new BankAccount[ MAX_ACCOUNTS ];
```

- Change the method name `setAccount` to `addAccount` and modify the implementation of the method accordingly! This method adds a new BankAccount if and only if the `accounts` array has less than MAX_ACCOUNTS elements.

```
public void addAccount( BankAccount account)
```

- Add an `accountNumber` parameter to the `getAccount` method. Search the given `accountNumber` in the `accounts` array and return the desired account. If the desired account does not exist, return `null`!

```
public BankAccount getAccount(String accountNumber)
```

OOP
Lab 3.

Note! Use the `equals` method for Strings comparison!

```
String string1 = "apple";
String string2 = "applex";
System.out.println( string1.equals(string2) );
```

- Create a getter method for the field numAccounts.

```
public int getNumAccounts()
```

- Modify the `closeAccount` method. Add an `accountNumber` parameter to the method and delete the desired account from the array. Be careful when you delete an element from an array. The array should not contain holes (null values). If the deletion is successful, decrease the `numAccounts`. If the desired `accountNumber` does not exist, print an error message!

- Modify the `toString` method in order to print besides the customer first and last name all information about the owned accounts.
  Example:

```
John BLACK accounts:
      BankAccount{accountNumber='OTP00001', balance=10000.0}
      BankAccount{accountNumber='OTP00002', balance=0.0}
```

  Use the `StringBuffer` type instead of `String`!

```
@Override
public String toString() {
   StringBuffer  result =  new StringBuffer();
   result.append(firstName + ' ' + lastName + " accounts:\n");
   for(int i=0; i<numAccounts; ++i){
       result.append( "\t" + accounts[i] +"\n");
   }
   return result.toString();
}
```

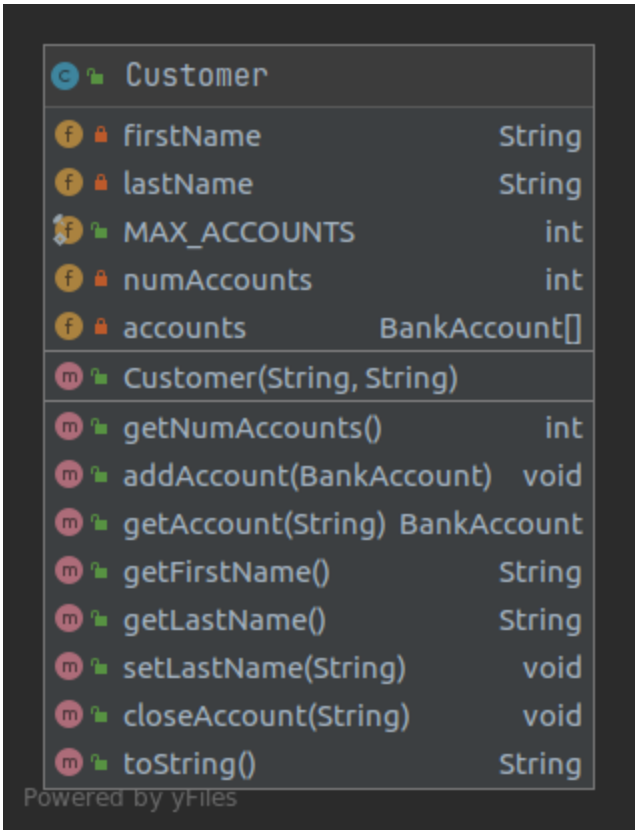

Fig.4. Modified Customer class.

- Test your class!
    - Create at least 2 customers. Add 5 bank accounts to the first customer and 9 bank accounts to the second customer.
    - Print the customers.
    - Deposit in each account a random amount of money.
    - Close the first account of the first customer.
    - Close the last account of the second customer.
    - Print the customers.