Lab: Defining Classes

Problems for exercises and homework for the "Java OOP Basics" course @ SoftUni.

You can check your solutions here: https://judge.softuni.bg/Contests/Practice/Index/474#0.

Part I: Defining Classes

1. Define Bank Account Class

Create a class named **BankAccount**.

The class should have public fields for:

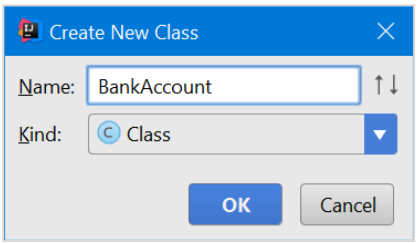
- Id: int
- Balance: double

You should be able to use the class like this:

```
public class Main {
    public static void main(String[] args) {

        BankAccount acc = new BankAccount();

        acc.id = 1;
        acc.balance = 15;

        System.out.printf(
                "Account ID%d, balance %.2f",
                acc.id,
                acc.balance
        );
    }
}
```

Solution

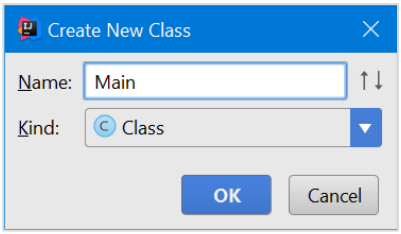
Create a **new class** and ensure **proper naming**



Define the fields

```
package p01_define_bank_account;

public class BankAccount {
    public int id;
    public double balance;
}
```

Create a Test client in the same package



You should be now able to use your class

```
public class Main {
    public static void main(String[] args) {

        BankAccount acc = new BankAccount();

        acc.id = 1;
        acc.balance = 15;

        System.out.printf(
                "Account ID%d, balance %.2f",
                acc.id,
                acc.balance
        );
    }
}
```

2. Getters and Setters

Create a class **BankAccount** (you can modify your previous implementation).

The class should have private fields for:

- Id: int
- Balance: double

Note: When declaring the fields, consider using primitive data types.

The class should also have public methods for:

- **`setId(int id): void`**
- **`getBalance(): double`**
- **`deposit(double amount): void`**
- **`withdraw(double amount): void`**

Override method **`toString()`**.

You should be able to use the class like this:

```
BankAccount acc = new BankAccount();

acc.setId(1);
acc.deposit(15);
acc.withdraw(5);

System.out.printf(
        "Account %s, balance %.2f",
        acc,
        acc.getBalance()
);
```

Solution

Create the class as usual or **use and modify** your previous implementation.

```
public class BankAccount {

}
```

Make **all fields** of the class **private**

```
public class BankAccount {

    private int id;
    private double balance;
}
```

Create a **setter** for the id

```
public void setId(int id) {
    this.id = id;
}
```

Create a **getter** for the balance

```
public double getBalance() {
    return this.balance;
}
```

Create a method **deposit(double amount)**

```
public void deposit(double amount) {
    // TODO: consider negative amount value
    this.balance += amount;
}
```

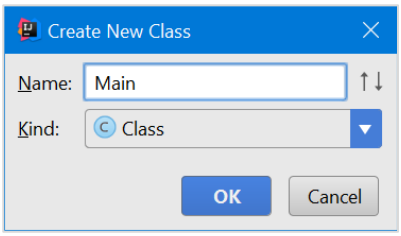
Create a method **withdraw(double amount)**

```
public void withdraw(double amount) {
    // TODO: consider insufficient balance
    this.balance -= amount;
}
```

Override method **toString()**

```
@Override
public String toString() {
    return "ID" + this.id;
}
```

Create a Test client in the same package



You should be able to use the class like this:

```
BankAccount acc = new BankAccount();

acc.setId(1);
acc.deposit(15);
acc.withdraw(5);

System.out.printf(
        "Account %s, balance %.2f",
        acc,
        acc.getBalance()
);
```

3. Test Client

Create a test client that tests your **BankAccount** class.

Support the **following commands**:

- **Create {Id}**
- **Deposit {Id} {Amount}**
- **Withdraw {Id} {Amount}**
- **Print {Id}**
- **End**

If you try to create an account with existing Id, print **"Account already exists"**.

If you try to perform an operation on **non-existing account** with existing Id, print **"Account does not exist"**.

If you try to withdraw an amount larger than the balance, print **"Insufficient balance"**.

Print command should print **"Account ID{id}, balance {balance}"**. Round the balance to the second digit after the decimal separator.

Examples

Input	Output
Create 1 Create 1 Deposit 1 20 Withdraw 1 30 Withdraw 1 10 Print 1 End	Account already exists Insufficient balance Account ID1, balance 10.00
Create 1 Deposit 2 20 Withdraw 2 30 Print 2 End	Account does not exist Account does not exist Account does not exist

Solution

Create a **HashMap<Integer, BankAccount>** to store existing accounts

```
HashMap<Integer, BankAccount> accounts = new HashMap<>();
```

Create the input loop

```
Scanner scanner = new Scanner(System.in);
String command = scanner.nextLine();
while (!command.equals("End")) {
    String[] cmdArgs = command.split("\\s+");

    command = scanner.nextLine();
}
```

Check the **type of command** and **execute** accordingly (***optional:*** *you can create a separate method for each command*)

```
String cmdType = cmdArgs[0];
switch (cmdType) {
    case "Create":
        execCreate(cmdArgs, accounts);
        break;
    case "Deposit":
        execDeposit(cmdArgs, accounts);
        break;
    case "Withdraw":
        execWithdraw(cmdArgs, accounts);
        break;
    case "Print":
        execPrint(cmdArgs, accounts);
        break;
}
```

Implement the Create command

```
int id = Integer.valueOf(cmdArgs[1]);
if (accounts.containsKey(id)) {
    System.out.println("Account already exists");
} else {
    BankAccount account = new BankAccount();
    account.setId(id);
    accounts.put(id, account);
}
```

Implement the rest of the commands following the same logic

Part II: Constructors and Static Members

4. Define Person Class

Create a Person class.

The class should have **private fields** for:

- Name: String
- Age: int
- Accounts: **`List<BankAccount>`**

The class should have **constructors**:

- **`Person(String name, int age)`**
- **`Person(String name, int age, List<BankAccount> accounts)`**

The class should also have **public methods** for:

- **`getBalance(): double`**

You should be able to use the class like this:

Solution

Create the class as usual

```
public class Person {

    private String name;
    private int age;
    private List<BankAccount> accounts;
}
```

Create a constructor with two parameters

```
public Person(String name, int age) {
    this.name = name;
    this.age = age;
    this.accounts = new ArrayList<>();
}
```

Create a constructor with three parameters

```
public Person(String name, int age, List<BankAccount> accounts) {
    this.name = name;
    this.age = age;
    this.accounts = accounts;
}
```

Create method **getBalance()**

```
public double getBalance() {
    return this.accounts.stream().mapToDouble(x -> x.getBalance()).sum();
}
```

Optional: You can take advantage of **constructor chaining**

```
public Person(String name, int age) {
    this(name, age, new ArrayList<>());
}

public Person(String name, int age, List<BankAccount> accounts) {
    this.name = name;
    this.age = age;
    this.accounts = accounts;
}
```

5. Static Id and Rate

Create class **BankAccount**.

The class should have **private fields** for:

- Id: int (Starts from 1 and increments for every new account)
- Balance: double
- Interest rate: double (Shared for all accounts. Default value: 0.02)

The class should also have public methods for:

- **setInterestRate(double interest): void (static)**
- **getInterest(int Years): double**
- **deposit(double amount): void**

Create a test client supporting the following commands:

- **Create**
- **Deposit {Id} {Amount}**
- **SetInterest {Interest}**
- **GetInterest {ID} {Years}**
- **End**

Examples

Input	Output	Comments
Create Deposit 1 20 GetInterest 1 10 End	Account ID1 created Deposited 20 to ID1 4.00	
Create	Account ID1 created	Sets the global interest rate to 1.

Create Deposit 1 20 Deposit 3 20 Deposit 2 10 SetInterest 1 GetInterest 1 1 GetInterest 2 1 GetInterest 3 1 End	Account ID2 created Deposited 20 to ID1 Account does not exist Deposited 10 to ID2 20.00 10.00 Account does not exist	Prints interest for bank account with id 1 for 1 year period.

Solution

Create the class as usual and create a constant for the default interest rate

```
public class BankAccount {

    private final static double DEFAULT_INTEREST = 0.02;
}
```

Create the static and non-static fields, **all private**

```
public class BankAccount {

    private final static double DEFAULT_INTEREST = 0.02;

    private static double rate = DEFAULT_INTEREST;
    private static int bankAccountCount;

    private int id;
    private double balance;
}
```

Set the id of an account upon creation while incrementing the global account count

```
public BankAccount() {
    this.id = ++bankAccountCount;
}
```

Create a setter for the global interest rate. Making the method static will let you access it through the class name

```
public static void setInterest(double interest) {
    rate = interest;
}
```

Implement **deposit()** and **getInterest()**

```
public void deposit(double amount) {
    this.balance += amount;
}

public double getInterest(int years) {
    return this.balance * rate * years;
}
```

Override **toString()** method

```
@Override
public String toString() {
    return "ID" + this.id;
}
```

Create a Test client in the same package