# Lab: Defining Classes

Problems for exercises and homework for the ["Java DB Fundamentals" course @ SoftUni](https://softuni.bg/courses/databases-advanced-hibernate).

# Part I: Defining Classes

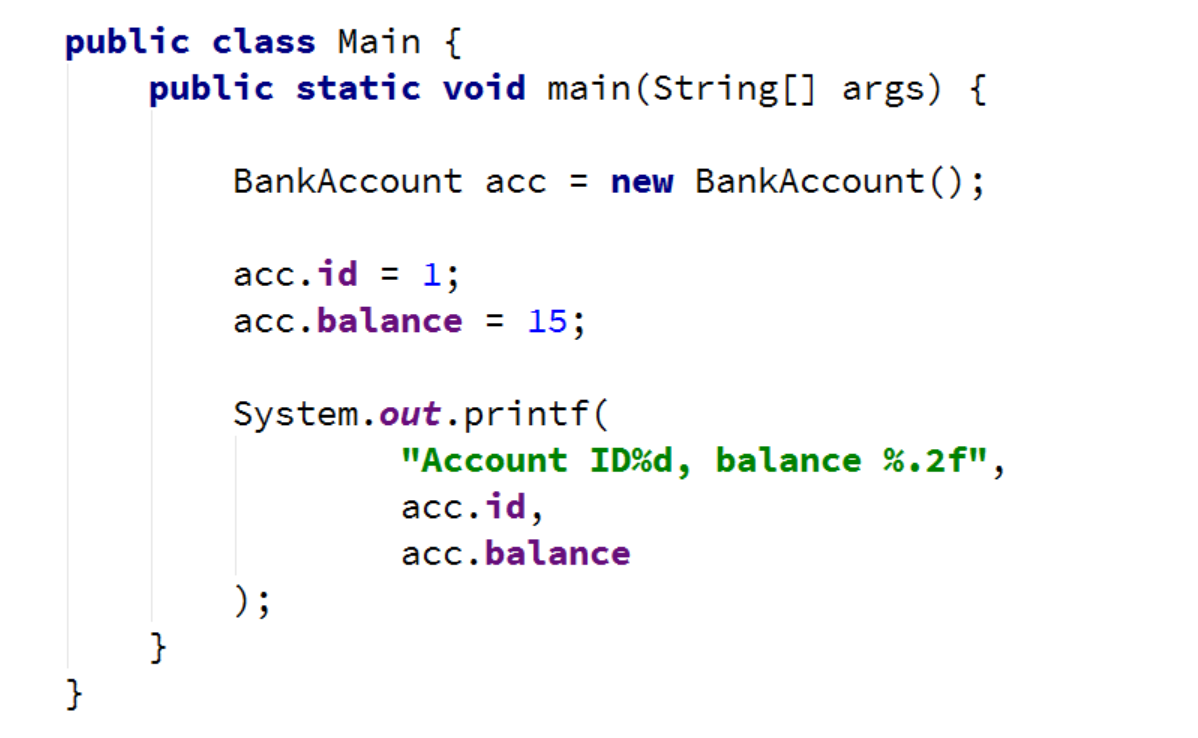
## Define Bank Account Class

Create a class named BankAccount.

The class should have public fields for:

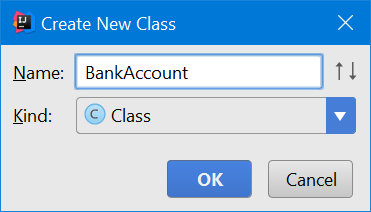
* Id: Integer
* Balance: Double

You should be able to use the class like this:

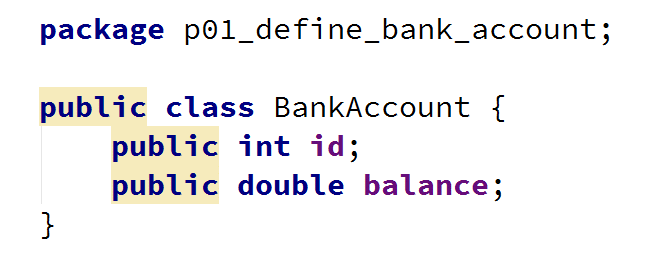


### Solution

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



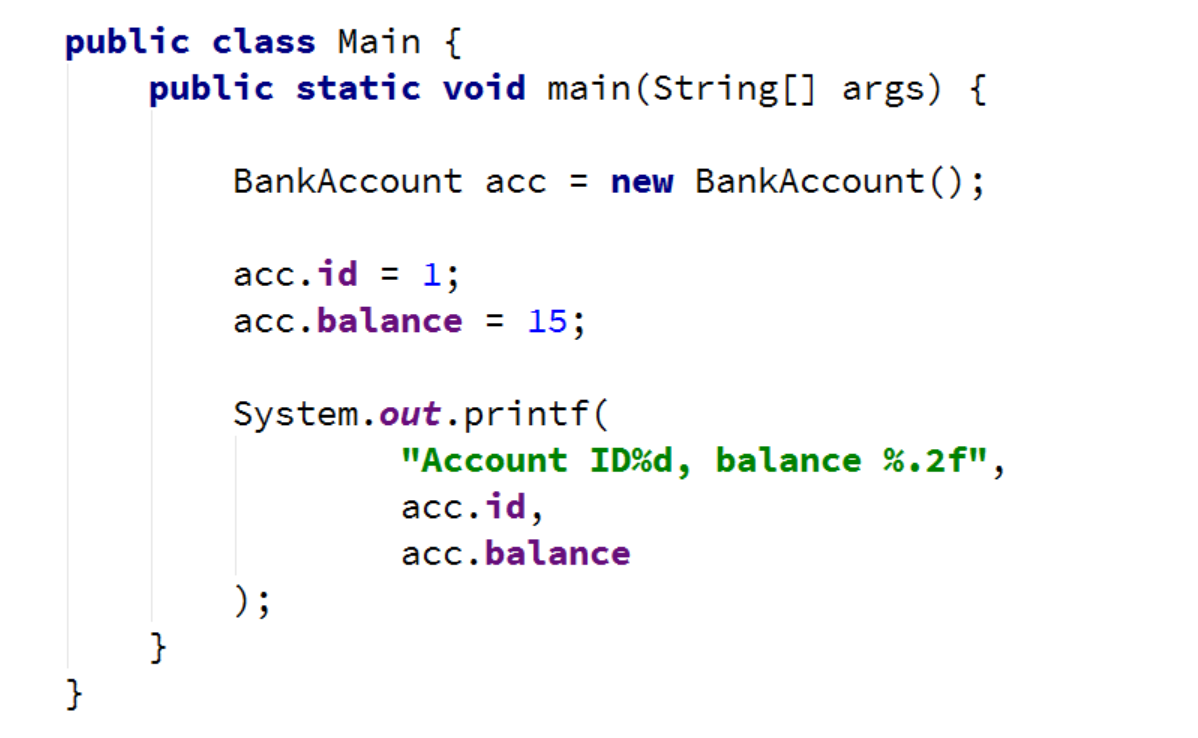
Define the fields



Create a Test client in the same package



You should be now able to use your class



## Getters and Setters

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

The class should have private fields for:

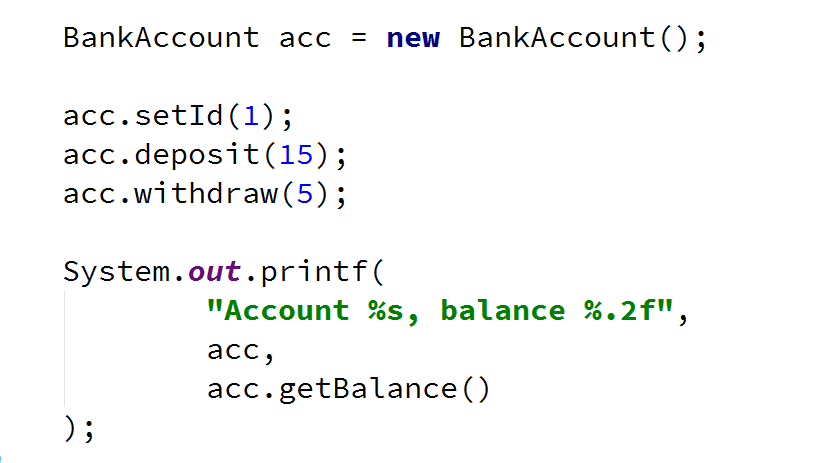
* Id: Integer
* Balance: Double

The class should also have public methods for:

* setId(): void
* getBalance(): Double
* deposit(Double amount): void
* withdraw(Double amount): void

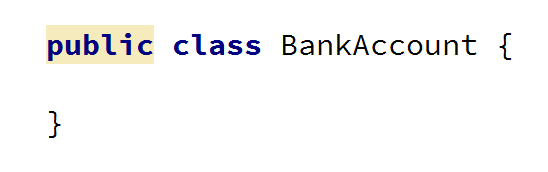
Override method toString().

You should be able to use the class like this:

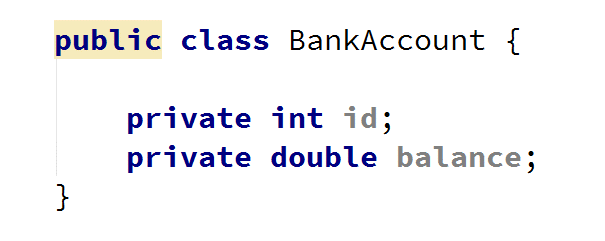


### Solution

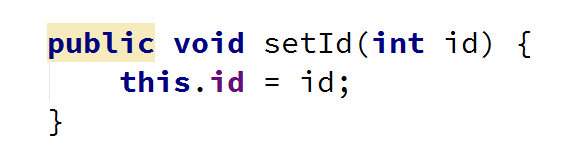
**Create the class** as usual or **use and modify** your previous implementation.



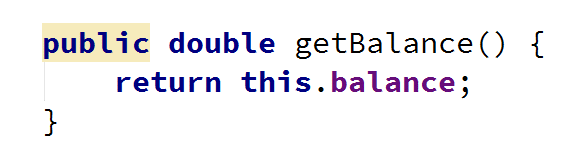
Make **all fields** of the class **private**



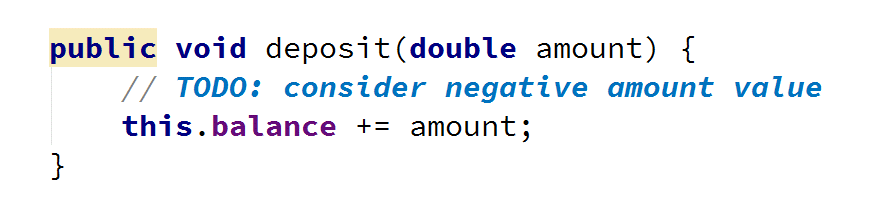
Create a **setter** for the id



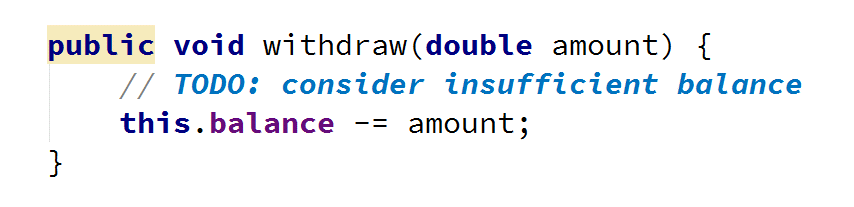
Create a **getter** for the balance



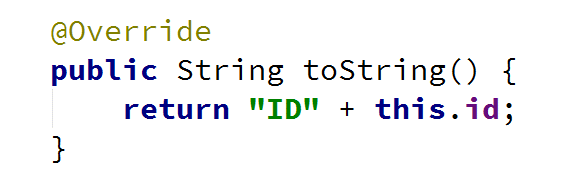
Create a method deposit(double amount)



Create a method withdraw(double amount)



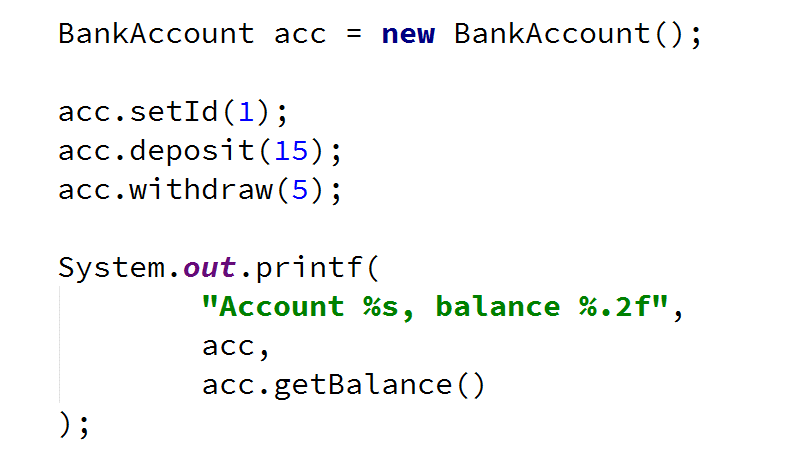
Override method toString()



Create a Test client in the same package



You should be able to use the class like this:



## 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"**.

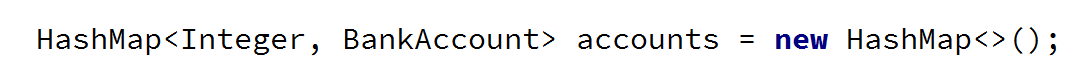
Print command should print **"Account ID{id}, balance {balance}"**. Round the balance to the second digit after the decimal separator.

### Examples

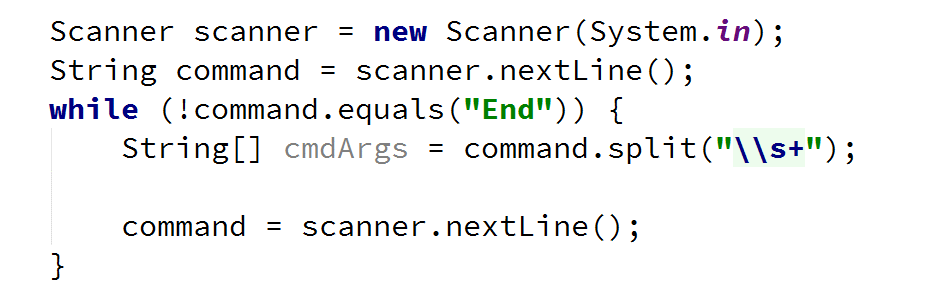
|  |  |
| --- | --- |
| **Input** | **Output** |
| Create 1  Create 1  Deposit 1 20  Withdraw 1 30  Withdraw 1 10  Print 1  End | Account already exists  Insufficient balance  Account ID1, balance 10.00 |
| Create 1  Deposit 2 20  Withdraw 2 30  Print 2  End | Account does not exist  Account does not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

### Solution

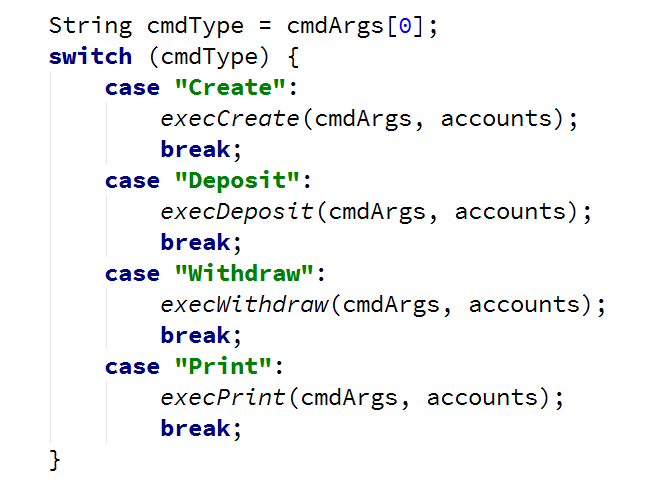
Create a HashMap<Integer, BankAccount> to store existing accounts



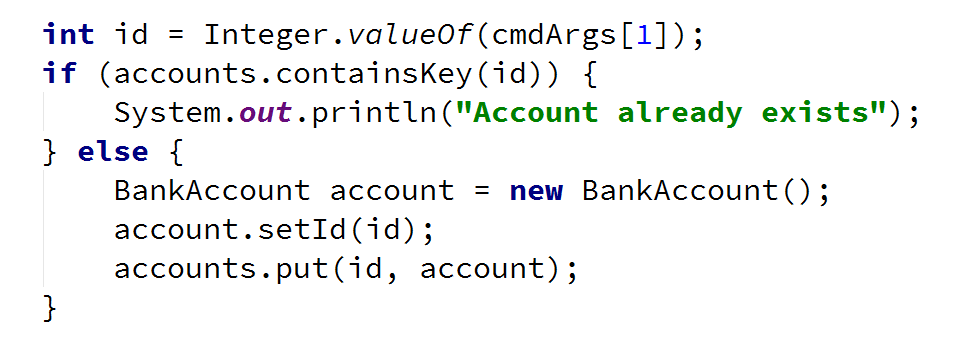
Create the input loop



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



Implement the Create command



Implement the rest of the commands following the same logic

# Part II: Constructors and Static Members

## Define Person Class

Create a Person class.

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

* Name: String
* Age: Integer
* Accounts: List<BankAccount>

The class should have **constructors**:

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

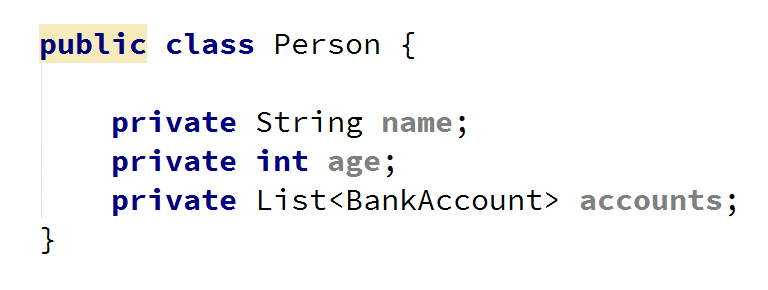
The class should also have **public methods** for:

* getBalance(): void

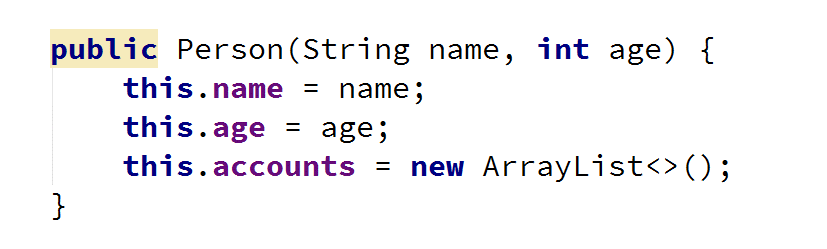
You should be able to use the class like this:

### Solution

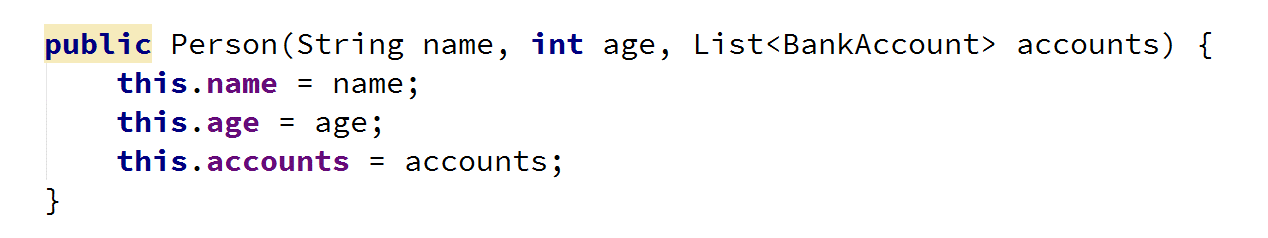
Create the class as usual



Create a constructor with two parameters



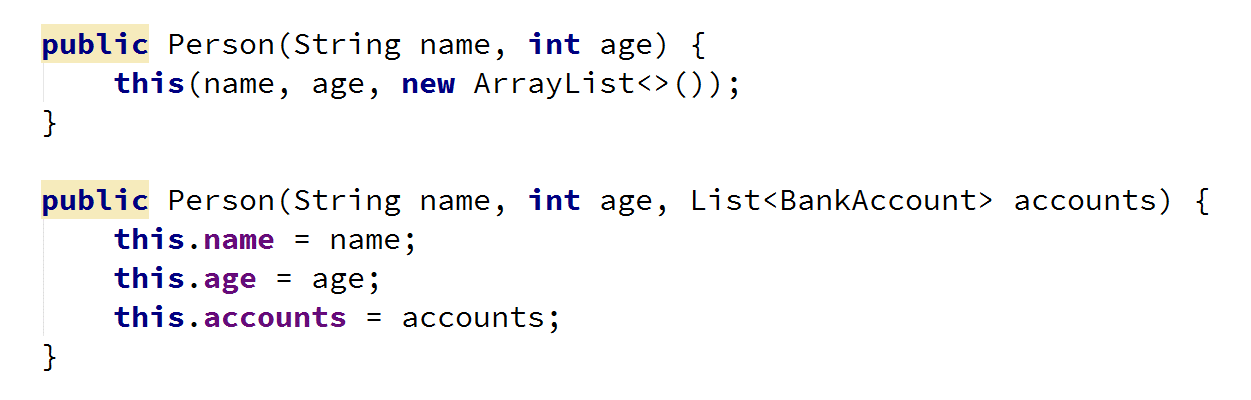
Create a constructor with three parameters



Create method getBalance()



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



## Static Id and Rate

Create class BankAccount.

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

* Id: Integer (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(Integer Years): Double
* deposit(Double amount): void

Create a test client supporting the following commands:

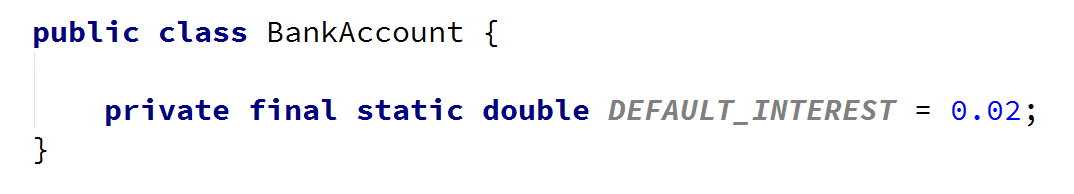
* **Create**
* **Deposit {Id} {Amount}**
* **SetInterest {Interest}**
* **GetInterest {ID} {Years}**
* **End**

### Examples

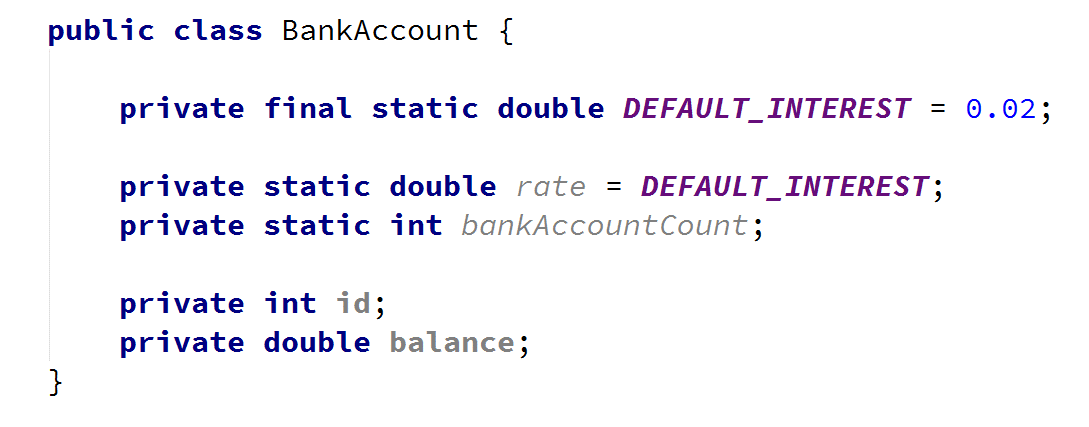
|  |  |
| --- | --- |
| **Input** | **Output** |
| Create  Deposit 1 20  GetInterest 1 10  End | Account ID1 created  Deposited 20 to ID1  4.00 |
| Create  Create  Deposit 1 20  Deposit 3 20  Deposit 2 10  SetInterest 1  GetInterest 1 1  GetInterest 2 1  GetInterest 3 1  End | Account ID1 created  Account ID2 created  Deposited 20 to ID1  Account does not exist  Deposited 10 to ID2  20.00  10.00  Account does not exist |

### Solution

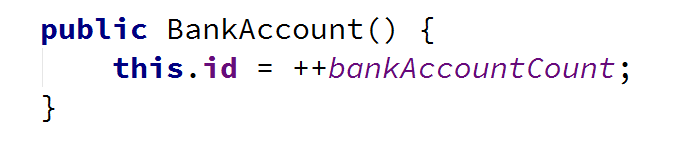
Create the class as usual and create a constant for the default interest rate



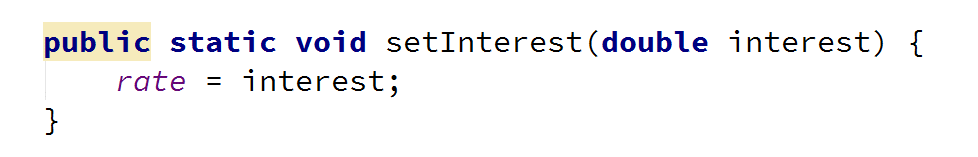
Create the static and non-static fields, **all private**



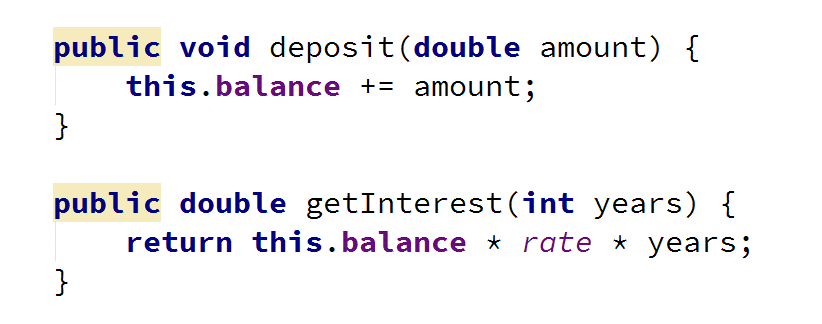
Set the id of an account upon creation while incrementing the global account count



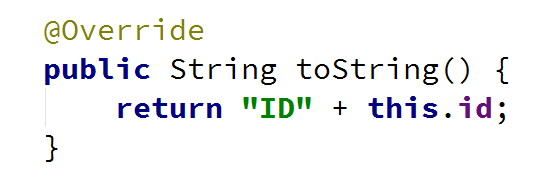
Create a setter for the global interest rate. Making the method static will let you access it through the class name



Implement deposit() and getInterest()



Override toString() method



Create a Test client in the same package



Start by