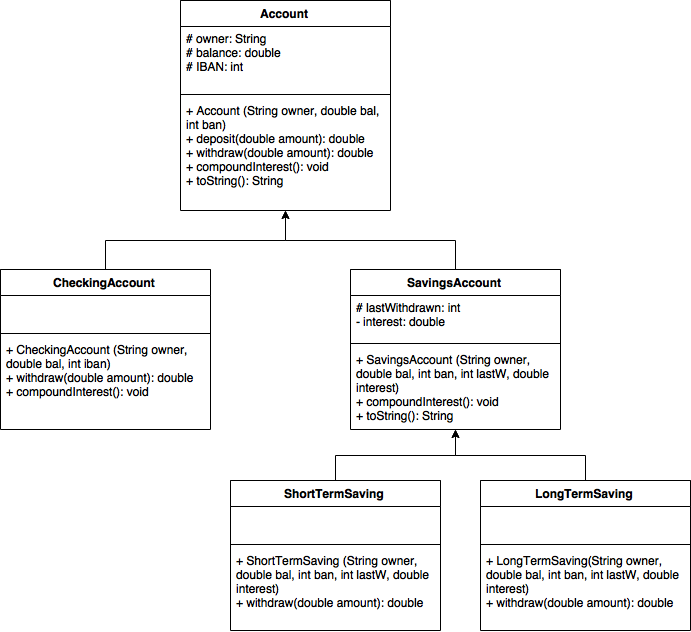
|  |  |  |
| --- | --- | --- |
| **KING SAUD UNIVERSITY**  **COLLEGE OF COMPUTER AND INFORMATION SCIENCES Computer Science Department** | | |
| **CSC 113: Introduction to Programming II** | **Lab\_Sheet#5** | **2nd Semester 1438** |



Given the above UML diagram, write the complete java implementation for all the classes according to the following description – some classes are Abstract, you need to find out which ones they are:

**Account**

1. Attributes

* **owner:** the name of the account owner.
* **balance:** the amount of money currently in the bank account.
* **IBAN:** the account number.

1. Methods

* **Account (owner: String, bal: double, iban: int) >>** constructor to initialize Account attributes
* **deposit (amount: double): double >>** adds the received amount to the balance and returns the value of the new balance
* **withdraw (amount: double): double >>** an abstract method that deducts money depending on account type
* **compoundInterest (): void >>** an abstract method that applies interest to balance
* **toString (): String >>** returns a string representation of Account object

**CheckingAccount**

1. Methods

* **CheckingAccount (owner: String, bal: double, iban: int) >>** constructor to initialize CheckingAccount attributes
* **withdraw (amount: double): double >>** method that deducts amount from balance and returns the value new balance
* **compoundInterest (): void >>** interest is 0 for a checking account, so it simply prints the current balance

**SavingsAccount**

1. Attributes

* **lastWithdrawn:** the number of days since the account was last withdrawn from.
* **interest:** the percentage by which the balance increases when it is compounded

1. Methods

* **SavingsAccount (owner: String, bal: double, iban: int, interest: double) >>** constructor to initialize SavingsAccount attributes
* **compoundInterest (): void >>** increases the balance by the percentage in interest (e.g. if balance was 100 and interest was 5% then new balance will be 105) then prints new balance
* **toString (): String >>** returns a string representation of SavingsAccount object

**ShortTermSaving**

1. Methods

* **ShortTermSaving (owner: String, bal: double, iban: int, interest: double) >>** constructor to initialize ShortTermSaving attributes
* **withdraw (amount: double): double >>** method that deducts amount from balance and prints the value new balance only if the last time withdrawn was more than 30 days ago, otherwise withdrawal is not allowed. Remember to update lastWithdrawn to 0.

**LongTermSaving**

1. Methods

* **LongTermSaving (owner: String, bal: double, iban: int, interest: double) >>** constructor to initialize LongTermSaving attributes
* **withdraw (amount: double): double >>** method that deducts amount from balance and prints the value new balance only if the last time withdrawn was more than 365 days ago, otherwise withdrawal is not allowed. Remember to update lastWithdrawn to 0.

Finally, implement **TestAccount** class with **main** method to do the following tasks:

1. Create an array of Accounts of size 6.
2. Initialize the objects of the array with the following objects:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Owner | Balance | IBAN | lastWithdrawn | interest |
| ShortTerm Saving | Khalid Ali | 1500.0 | 23521 | 5 | 8% |
| Checking | Sarah Sami | 352.0 | 23523 |  |  |
| ShortTerm Saving | Mohammed  Waleed | 2333.0 | 3452 | 50 | 5% |
| LongTerm Saving | Noura Ibrahim | 24000.5 | 45425 | 400 | 25% |

1. Print the information of all accounts
2. Deposit 400 Riyals to all accounts and print new balance
3. Withdraw 50 Riyals from all accounts and print new balance
4. Compound interest for all accounts
5. Print the information of all accounts after the changes