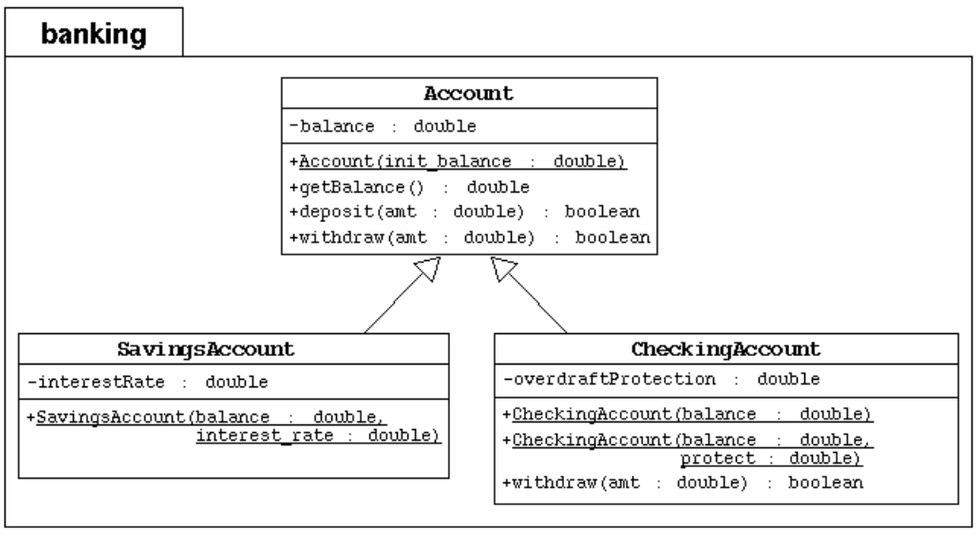
Computing Problem Number *2*
Computing Problem Title *Creating Subclasses of Bank Accounts*

Objective

In this exercise you will explore the purpose of applying *inheritance* given an existing class design. You will use the existing class as a superclass and create subclasses by inheriting all attributes and methods from that superclass and add additional attributes and/or methods to these subclasses for it to be used effectively.

Directions

Version 1: Implementing the Account Subclasses



Start by changing your working directory to `.../Computing Problem 2/version1` on your computer.

To the `banking` package, you will add the `SavingsAccount` and `CheckingAccount` subclasses as modeled by the UML diagram above.

To include the `SavingsAccount` and `CheckingAccount` subclasses in the package during compilation, you should include the code `package banking;` at the topmost part of your .java file.

The Savings Account Subclass

1. Implement the `SavingsAccount` class as modeled in the above UML diagram.
2. The `SavingsAccount` class must extend the `Account` class.
3. It must include an `interestRate` attribute with type `double`.
4. It must include a public constructor that takes two parameters: `balance` and `interest_rate`. This constructor must pass the `balance` parameter to the parent constructor by calling `super(balance)`.

The Checking Account Subclass

1. Implement the `CheckingAccount` class as modeled in the above UML diagram.
2. The `CheckingAccount` class must extend the `Account` class.
3. It must include an `overdraftProtection` attribute with type `double`.
4. It must include one public constructor that takes one parameter: `balance`. This constructor must pass the `balance` parameter to the parent constructor by calling `super(balance)`.
5. It must include another public constructor that takes two parameters: `balance` and `protect`. This constructor must pass the `balance` parameter to the parent constructor by calling `super(balance)` and set the `overdraftProtection` attribute by the value of `protect`.
6. The `CheckingAccount` class must override the `withdraw` method. It must perform the following check: if the current balance is adequate to cover the `amount` to withdraw, then proceed as usual. If not *and if there is overdraft protection*, then attempt to cover the difference (`balance - amount`) by value of the

`overdraftProtection`. If the amount needed to cover the overdraft is greater than the current level of protection, then the whole transaction must fail with the checking balance unaffected.

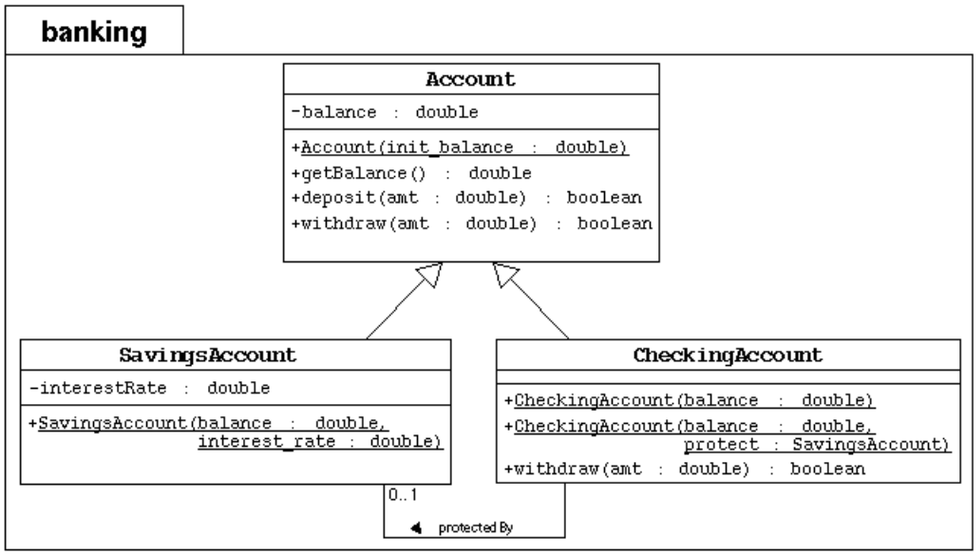
Test the Code

In the main directory (`.../Computing Problem 2/version1`), compile and execute the `TestBanking` program. The output should be:

```
Creating the customer Jane Smith.
Creating her Savings Account with a 500.00 balance and 3% interest.
Creating the customer Owen Bryant.
Creating his Checking Account with a 500.00 balance and no overdraft protection.
Creating the customer Tim Soley.
Creating his Checking Account with a 500.00 balance and 500.00 in overdraft
protection.
Creating the customer Maria Soley.
Maria shares her Checking Account with her husband Tim.

Retrieving the customer Jane Smith with her savings account.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Simms, Jane] has a balance of 324.88

Retrieving the customer Owen Bryant with his checking account with no overdraft
protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: false
Customer [Bryant, Owen] has a balance of 324.88

Retrieving the customer Tim Soley with his checking account that has overdraft
protection.
Withdraw 150.00: true
Deposit 22.50: true
Withdraw 47.62: true
Withdraw 400.00: true
Customer [Soley, Tim] has a balance of 0.0

Retrieving the customer Maria Soley with her joint checking account with husband
Tim.
Deposit 150.00: true
Withdraw 750.00: false
Customer [Soley, Maria] has a balance of 150.0
```

Notice that Jane's savings account and Owen's checking account fundamentally behave as a plain-old bank account. But Tim & Maria's joint checking account has 500.00 worth of overdraft protection. Tim's transactions dip into that protection and therefore his ending balance is 0.00. His account's overdraft protection level is 424.88. Finally, Maria deposits 150.00 into this joint account; raising the balance from 0.00 to 150.00. Then she tries withdraw 1000.00, which fails because neither the balance nor the overdraft protection can cover that requested amount.

Version 2: Checking Account protected by a Savings Account

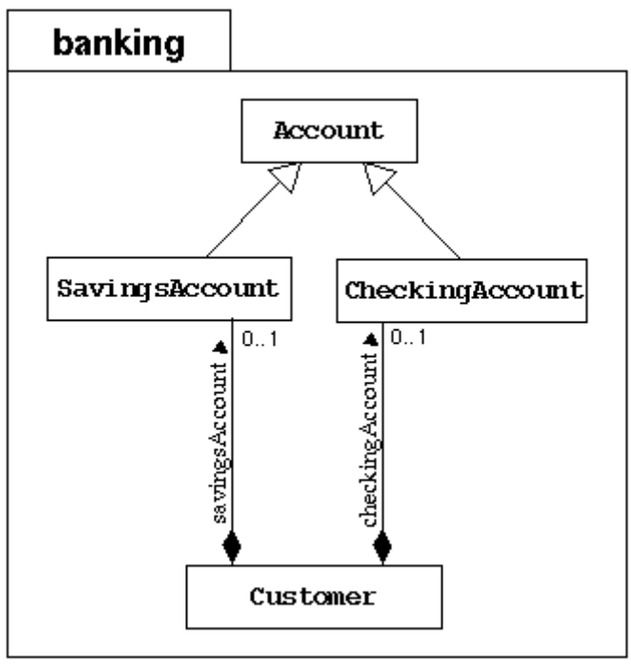


Start by changing your working directory to `.../Computing Problem 2/version2` on your computer. Copy the Version 1 banking project files (from `.../Computing Problem 2/version1`) in this package directory.

Modify the Checking Account Class

1. Remove the attribute `overdraftProtection` on the `CheckingAccount` class, but a *private* association attribute, called `protectedBy` must be added with the type `SavingsAccount`.
2. Reimplement the second constructor with the same signature shown in the UML and re-implement its assignment to the newly created attribute, `protectedBy`.
3. Re-implement the overridden method, `withdraw`, such that instead of `overdraftProtection`, use the `protectedBy` — `SavingsAccount` object instead.

Modifying Customer to Hold Two Accounts



Modify the `Customer` class to hold two bank accounts: one for savings and one for checking; both are optional.

1. Previously, the `Customer` class contained an association attribute called `account` to hold an `Account` object. Rewrite this class to contain two *private* association attributes: `savingsAccount` and `checkingAccount`.
2. Remove the `getAccount` *accessor* method and include two *accessor* methods: `getSavingsAccount` and `getCheckingAccount`, which returns the savings and checking accounts, respectively.

3. Remove the `setAccount` *mutator* method and include two *mutator* methods: `setSavingsAccount` and `setCheckingAccount`, which set the savings and checking account associations, respectively.

Test the Code

In the main `Problem Set 6/Problem 3` directory, compile and execute the `TestBanking` program. The output should be:

```
Customer [Simms, Jane] has a checking balance of 200.0 and a savings balance of
500.0
Checking Acct [Simms, Jane]: withdraw 150.00 succeeds? true
Checking Acct [Simms, Jane]: deposit 22.50 succeeds? true
Checking Acct [Simms, Jane]: withdraw 147.62 succeeds? true
Customer [Simms, Jane] has a checking balance of 0.0 and a savings balance of
424.88

Customer [Bryant, Owen] has a checking balance of 200.0
Checking Acct [Bryant, Owen]: withdraw 100.00 succeeds? true
Checking Acct [Bryant, Owen]: deposit 25.00 succeeds? true
Checking Acct [Bryant, Owen]: withdraw 175.00 succeeds? false
Customer [Bryant, Owen] has a checking balance of 125.0
```

Notice that Jane's checking account is protected by her savings account in the last transaction; whereas, Owen has no overdraft protection, so the last transaction on his account fails and the balance is not affected.