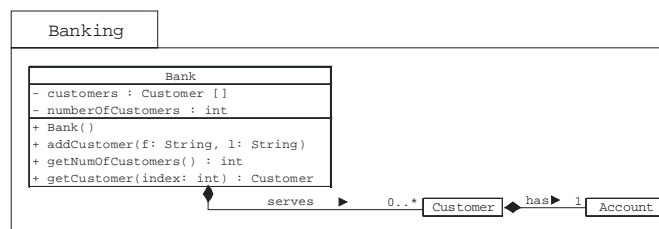


PRAKTIKUM 2 PEMROGRAMAN BERORIENTASI OBYEK II

Praktikum 2.1

1. Deklarasikan sebuah variabel yang dinamakan `matrix` dengan tipe `int[][]` (array dari array `int`). Inisialisasikan matriks array tersebut dengan sebuah array dari lima array.
2. Populasikan setiap array yang di dalam (*inner array*) dengan cara sebagai berikut: lakukan loop dalam `matrix` dari nol hingga panjang array tersebut, gunakan `i` sebagai index. Dalam setiap iterasi, berikan nilai pada `matrix[i]` dengan sebuah array integer baru yang berukuran `i`. Lalu, lakukan proses loop pada setiap elemen dari array tersebut, dengan menggunakan variabel index `j`. Dalam setiap kali iterasi, berikan nilai `matrix[i][j]` dengan nilai $(i * j)$.
3. Cetak array `matrix` dengan melakukan iterasi pada array bagian luar (*outer array*) dan cetak setiap *inner array* pada baris yang terpisah. Kompilasikan program `TestArrays` dan jalankan. Apa yang akan terjadi ?

Praktikum 2.2



1. Buatlah sebuah direktori banking. Salin file project Banking dari tugas sebelumnya ke dalam direktori banking tersebut.

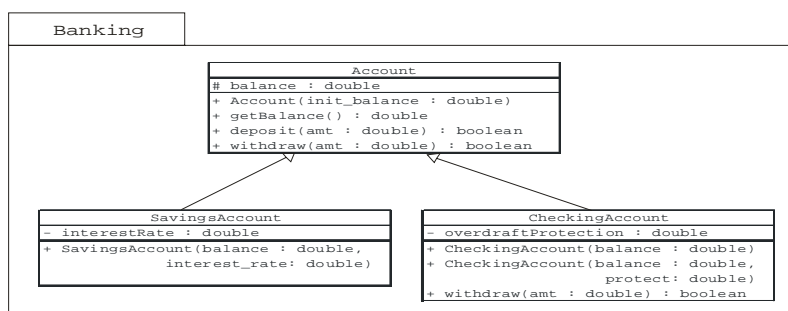
Modifikasi pada kelas `Account`

2. Ubahlah method `deposit` sehingga method ini mengembalikan nilai `true` (yang berarti bahwa proses deposit berhasil).
3. Ubahlah method `withdraw` untuk mengecek bahwa jumlah uang (*amount/amt*) yang ditarik tidak lebih besar dari jumlah uang di *balance*. Jika *amt* lebih kecil dari *balance*, maka kurangi sejumlah *amount* dari *balance* dan kembalikan nilai `true`. Jika tidak, maka *balance* dibiarkan dan kembalikan nilai `false`.

Pembuatan kelas `Bank`

4. Tambahkan dua atribut pada kelas `Bank`: `customers` (array dari obyek `Customer`) dan `numberOfCustomers` (nilai integer yang menyatakan jumlah pelanggan).
5. Tambahkan sebuah konstruktor publik yang menginisialisasi array `customers` dengan satu ukuran maksimum tertentu (paling tidak lebih besar dari lima).
6. Tambahkan sebuah method `addCustomer`. Method ini akan membuat sebuah obyek `Customer` dari dua parameter (*first name* dan *last name*) dan meletakkannya dalam array `customer`. Method ini juga harus menambahkan nilai (*increment*) atribut `numberOfCustomers`.
7. Tambahkan sebuah method *accessor* `getNumOfCustomers`, yang akan mengembalikan atribut `numOfCustomers`.
8. Tambahkan method `getCustomer`. Method ini akan mengembalikan obyek `Customer` yang diasosiasikan dengan parameter *index* yang diberikan. Kompilasikan program `TestBanking` tersebut dan jalankan. Apa yang akan terjadi?

Praktikum 2.3



Mulailah dengan masuk ke direktori kerja anda.

Pada package `Banking`, tambahkan subclass `SavingsAccount` dan `CheckingAccount`, seperti tergambar di atas.

1. Buatlah sebuah direktori `banking`. Salinlah file proyek `Banking` dari proyek sebelumnya ke dalam direktori ini.

Modifikasi pada kelas `Account`

Pada diagram UML, kelas `Account` telah berubah. Atribut `balance` sekarang memiliki mode akses `protected` (diunjukkan dengan karakter `#`)

2. Ubahlah mode akses dari atribut `balance` menjadi `protected`

Modifikasi pada kelas `SavingsAccount`

Implementasikan kelas `SavingsAccount` sebagaimana tampak dalam gambar.

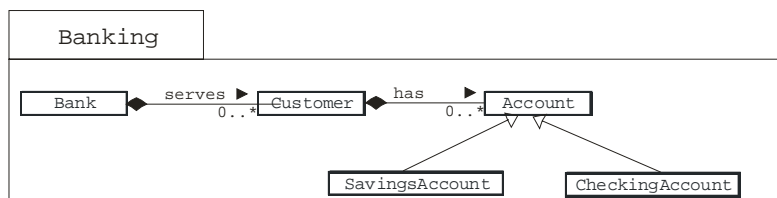
3. Kelas `SavingsAccount` haruslah meng-*extend* kelas `Account`.
4. Kelas tersebut juga harus memiliki atribut `interestRate` dengan tipe `double`.
5. Tambahkan pula konstruktor publik yang menggunakan dua parameter: `balance` dan `interest_rate`. Konstruktor ini akan melewati parameter `balance` ke dalam konstruktor induk dengan memanggil `super(balance)`.

Modifikasi pada kelas `CheckingAccount`

Implementasikan kelas `CheckingAccount` seperti tampak dalam gambar.

6. Kelas `CheckingAccount` tersebut haruslah meng-*extend* kelas `Account`.
7. Kelas tersebut juga harus memiliki atribut `overdraftProtection` dengan tipe `double`.
8. Tambahkan sebuah konstruktor publik yang menggunakan dua parameter: `balance` dan `protect`. Konstruktor ini haruslah melewati parameter `balance` pada konstruktor induk dengan memanggil `super(balance)` dan memberikan nilai pada atribut `overdraftProtection`.
9. Tambahkan pula sebuah konstruktor publik yang menggunakan sebuah parameter: `balance`. Konstruktor ini haruslah melewati parameter `balance` pada konstruktor yang lain (*overloaded constructor*) dengan pemanggilan `this`.
10. Kelas `CheckingAccount` harus meng-*override* method `withdraw`. Kelas ini juga harus melakukan pengecekan berikut: jika *current balance* mencukupi untuk proses penarikan (*withdraw*), maka lakukan proses penarikan biasa. Jika tidak dan jika ada *overdraft protection*, maka lakukan proses untuk menutupi selisih tersebut (`balance - amount`) dengan nilai `overdraftProtection`. Jika jumlah yang dibutuhkan untuk menutupi *overdraft* lebih besar dari level proteksi yang ada sekarang, maka batalkan seluruh proses transaksi tersebut.
11. Dalam direktori kerja, kompilasi dan jalankan program `TestBanking`. Apa yang akan terjadi ?

Praktikum 2.4



1. Buatlah sebuah direktori `banking`. Salin file project `Banking` dari tugas sebelumnya ke dalam direktori `banking` tersebut.
2. Modifikasikan kelas `Customer` sehingga kelas ini mampu memiliki banyak `account`. Kelas ini haruslah memiliki method publik berikut: `addAccount(Account)`, `getAccount(int)`, dan `getNumOfAccounts()`.

Penyelesaian pada program `TestBanking`

Program ini membuat satu set pelanggan dan `account`, lalu membuat sebuah laporan disertai dengan `account balance` mereka.

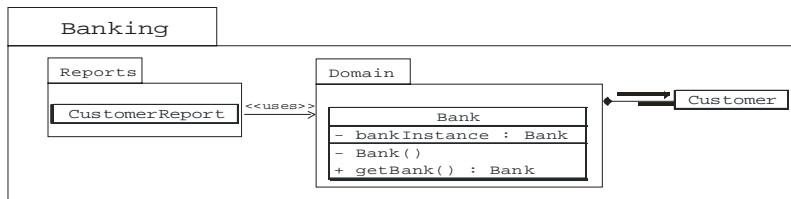
Di dalam file `TestBanking.java`, anda akan menemukan blok komentar yang berawal dan berakhir dengan `/** */`. Tempat ini menunjukkan lokasi di mana anda harus menyisipkan kode program.

3. Gunakan operator `instanceof` untuk menguji `account` jenis apa yang dimiliki dan set `account_type` ke dalam nilai yang sesuai, seperti `"Savings Account"` atau `"Checking Account"`.
4. Cetak jenis dari `account` dan `balance` nya. Gunakan pembuat format `currency_format` untuk membuat format mata uang yang sesuai.
5. Kompilasi dan jalankan program. Apa yang akan terjadi ?

Praktikum 2.5

Buatlah sebuah direktori banking. Salin file proyek Banking dari tugas sebelumnya ke dalam direktori banking/domain.

Dalam tahap ini, project Banking yang dibuat membutuhkan hirarki package yang lebih kompleks karena anda akan membuat sebuah kelas `CustomerReport` yang terdapat di dalam package `banking.reports`.



Modifikasi pada kelas `Bank` dengan mengimplementasi pola desain *Singleton*.

Mulai dengan masuk ke direktori kerja anda.

1. Modifikasikan kelas `Bank` dengan menambahkan method yang bertipe `public` dan `static`, `getBank`, yang mengembalikan *instance* dari kelas `Bank`.
2. Instans tunggal ini haruslah memiliki modifier `static` dan `private`. Buat pula konstruktor `Bank` menjadi `private`.

Modifikasi kelas `CustomerReport`

Dalam project `Banking` sebelumnya, laporan pelanggan (*customer report*) terdapat di dalam method `main` dari program `TestBanking`. Dalam latihan kali ini, kode tersebut ditempatkan di dalam kelas `CustomerReport` yang terdapat di dalam package `banking.reports`. Tugas anda adalah memodifikasi kelas ini sehingga ia menggunakan obyek bank *singleton*.

3. Cari baris kode yang ditandai dengan blok *comment* sebagai berikut: `/** */`. Ubah baris ini menjadi pengembalian berupa obyek bank *Singleton*.
4. Kompilasi dan jalankan program. Apa yang akan terjadi ?